The Dissertation Committee for Michael Shenoda Certifies that this is the approved version of the following dissertation:

Development of a Phase-by-Phase, Arrival-Based, Delay-Optimized Adaptive Traffic Signal Control Methodology with Metaheuristic Search

Committee:

Randy Machemehl, Supervisor

Loukas Kallivokas

David Morton

S. Travis Waller

Zhanmin Zhang

Development of a Phase-by-Phase, Arrival-Based, Delay-Optimized Adaptive Traffic Signal Control Methodology with Metaheuristic Search

by

Michael Shenoda, B.S.; M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2006

Dedication

This document is dedicated to my parents, who saw a son who was mired in complacent misery try to reach for distant personal satisfaction through dedicated hard work, and supported him with their kindness, generosity, patience, advice and, most importantly, love. Thanks for letting me be that son.

Acknowledgements

I would like to acknowledge the invaluable assistance of my advisor, Dr. Randy Machemehl, as well as the helpful input of all the members of my dissertation committee. I would also like to acknowledge the unwarranted aid and favors of my fellow students, and the insight and guidance provided by my instructors at The University of Texas at Austin. Finally, I would like to acknowledge the Southwest Region University Transportation Center, without whose generosity in support and the opportunity to work on Project No. 167243, "Adaptive Signal Control: Advancing the Concept", the completion this document would not have been possible.

Development of a Phase-by-Phase, Arrival-Based, Delay-Optimized Adaptive Traffic Signal Control Methodology with Metaheuristic Search

Publication No._____

Michael Shenoda, Ph.D.

The University of Texas at Austin, 2006

Supervisor: Randy Machemehl

Adaptive traffic signal control is the process by which the timing of a traffic signal is continuously adjusted based on the changing arrival patterns of vehicles at an intersection, usually with the goal of optimizing a given measure of effectiveness. Herein, a methodology is developed in which the characteristics of a traffic signal cycle are optimized at the conclusion of every phase based on the arrival times of vehicles to an intersection, using stopped delay as the measure of effectiveness. This optimization is solved using metaheuristic search procedures, namely tabu search, and embedded in an algorithm in which current vehicle arrival times are detected, arrival patterns over a specified horizon are predicted, the traffic signal timing is optimized, and the timings are sent to a traffic signal controller. The methodology is shown to provide improvement in performance for a number of intersection configurations and traffic regimes over traditional forms of traffic signal control, and the metaheuristic search is demonstrated to

significantly reduce the computation time for a solution as compared with other search procedures.

Table of Contents

List of Tables

List of Figures

Chapter 1: Definition and Objective of Research

A traffic intersection is defined by a point at which the sharing of right-of-way by two or more vehicles is required. In order to accomplish this sharing, intersection control is used. Contingent upon a number of warrants, as defined by the governing authority, a traffic signal may be used as an intersection control device. The traffic signal operates by allotting green time to the intersection approaches according to a chosen scheme or algorithm. The manner in which green time is allotted to these approaches has been the subject of much consideration and research. For purposes of discussion herein, three methods of distributing green time to approaches are considered (details on the first two methods and their implementation can be seen in Pignataro (1973) and May (1990)):

a) Pretimed: The traffic signal is set to provide a particular amount of green time to each approach during a cycle (generally defined as the provision of one green interval to each intersection approach). This length remains fixed for each approach for some period of time, whether that is an hour, a "rush hour" period, a few days, or indefinitely.
b) Actuated: The traffic signal provides a minimum length of green time to each approach during a cycle. This length may be incremented based on the vehicle arrival to the approach displaying green as observed by a detection device. The length of every green interval is also constrained by a maximum green time specification.
c) Adaptive: The traffic signal provides green time to each intersection approach based on anticipated arrivals for a cycle. Generally, as arrivals change from cycle to cycle, the length of green time provided to each approach is determined anew.

Pretimed traffic signal control is by far the most widely implemented method, and actuated control has also been widely implemented over the past few decades, particularly at isolated intersections. Research has delved deeply into these two methods with two main concerns:

a) to provide general improvements to the methods in order to enhance their overall performance
b) to assess their application in different situations, i.e. to determine which of the two methods is best suited to a particular intersection or network of intersections and how the chosen method can best be applied

Adaptive traffic signal control is a relatively new method; research began in the 1970's, has only recently been increasing, and even now, implementation is very sparse in North America. This is unfortunate, as adaptive control, given the proper attention, has the potential to diminish the need for constant adjustments to enhance performance, which is the concern of (a), and can replace both methods, since the signal can be programmed to act as one of the two or as its own signal control paradigm, which is the concern of (b). In all cases, the key area of potential is improved performance over some period of time, measured in the same terms described under the "pretimed" item above.

The objective of this research is broken down into two parts:

1. Generally, to "advance the concept" of adaptive traffic signal control. Within this, the structure of adaptive control operation will be deconstructed, and each particular process within the adaptive control procedure will be examined to determine where improvements are possible. This will be based on previously implemented adaptive control schemes, current research into the arena of adaptive

control, and concentration of research within each process upon the topics which might influence that process.

2. specifically, to develop a basic adaptive control scheme with the following goals:
    i. to be flexible to as many intersection and/or network configurations as may be encountered in the field
    ii. to be easily implementable on any intersection and/or network configuration with a minimum of adjustments specific to that configuration
    iii. to incorporate advancements as found in the first objective
    iv. to be transparent enough in structure to allow easy incorporation of future advancements in adaptive control in general or to specific processes
    v. to provide improved performance as compared with existing pretimed and actuated control methods, as well as comparable performance to existing adaptive control schemes

In essence, the objective is to investigate the shortcomings of existing adaptive control schemes by considering a general structure and analyzing processes within that structure, to suggest improvements, to create a scheme that implements the improvements, and to demonstrate the enhanced performance of that scheme.

Performance in the case of intersection control can be measured by a number of measures of effectiveness (MOEs) over the desired period of time [Pignataro (1973) and May (1990)]. These include the number of stops made by vehicles, the time vehicles are required to be stopped, or stopped delay (total or average per vehicle), the time vehicles spend in the system, or travel time, and the difference between the travel time and the minimum time that vehicles can spend in the system, or total delay. (Total delay may be

computed in a number of ways.) In static comparisons, different methods may prove to perform better when considering different MOEs. Therefore, an added goal of the developed adaptive control scheme will be to provide superior performance to pretimed or actuated for any given MOE, and that use of different MOEs in the adaptive control scheme will be facilitated.

With the nature of adaptive traffic signal control, as well as the purpose of further research into the topic, thus defined, an examination of the structure of adaptive control, as in Chapter 2, will yield an understanding of how the objectives can be reached.

Chapter 2: Structure of Adaptive Control

The adaptive control method has been defined under a number of models. These models will be referenced a number of times herein, particularly in terms of describing them and comparing them to the adaptive control scheme developed through this research. With regard to this particular research, however, the adaptive control method, despite the myriad models, can be defined through a basic structure. It is this structure that will be analyzed, with emphasis on enhancing its various processes. This structure includes:

- Detection: This is the process by which vehicles that enter a given approach to an intersection are recognized for processing by the adaptive control scheme. In question in this process are:

    a) the types of detection devices used – These may include loop detectors, laser or radar, and video
    b) the types of data measured – These may include time of arrival, speed, and axle spacing (and hence, vehicle type).
    c) the positioning of the devices used – This would involve investigating the distance for advance detection, number of detectors to be used, and use of detector arrays.

- Prediction: This is the process by which the data from the detection process are used to determine arrival patterns to be used by the adaptive control scheme. The scheme cannot process vehicles in "real time"; it can only make control determinations for vehicles over a given time horizon. Prediction creates a pattern

of vehicles over that time horizon for use in the control scheme. Some of the issues to be confronted in this process are:

a) length of time horizon to be used – This may be as short as a minimum vehicle headway (e.g. 2 seconds) and may be considered as a given cycle length, a 15-minute interval, a peak period, or may be left variable.
b) use of multiple time horizons – The horizons may have different lengths, and may be used for different purposes, such as decision thresholds and detection intervals.
c) process by which pattern is generated – This can include replicating a historically detected pattern, scaling a historical pattern, fitting a probability distribution, or using a time-series model.

- Optimization: This is the process by which the predicted vehicle arrivals are used to distribute green times to the various approaches of the intersection. In essence, this process optimizes a measure of effectiveness based on the vehicle arrivals. Among the issues involved with this process are:

a) measure of effectiveness to be used – As discussed earlier, this can be stopped delay, total delay, number of stops, or other measures, depending on the desire of the user.
b) objective function – This is the actual mathematical relationship that will determine the impact the vehicle arrival patterns have on the measure of effectiveness, and thus the green time distribution.

c) other factors – These can include other data based on the detection capabilities, as well as any others that may be deemed significant by the user; these may be assigned desired weights, as well.

Looking at adaptive control in terms of a well-defined structure such as that outlined above allows the analysis required to accomplish the aims of the methodology as described in Chapter 3.

Chpater 3: Aims of Adaptive Control

In general, traffic can be defined by demand regimes. These regimes represent the capacity of the roadway in question to process vehicles. They may be considered thusly:

- Low: ratio of volume of traffic to capacity of roadway is less than 50%
- Medium: ratio of volume of traffic to capacity of roadway is between 50 and 75%
- High: ratio of volume of traffic to capacity of roadway is greater than 75%

Demand regimes that represent more than 100% of the ratio above are considered oversaturated.

Adaptive control, as stated earlier, provides green signal indications to intersection approaches based on the anticipated arrivals on the approaches. Thus, in essence, it is the function of adaptive control to reduce as much as possible green time granted to approaches on which there are no vehicles approaching, while performing the opposite task to the approaches on which vehicles are approaching. Based on the possible demand regimes that can exist on these approaches, the benefits of adaptive control as compared to pretimed control will vary. When the demand is low on all approaches, green time can be distributed to vehicles as it is anticipated they will arrive. When demand is low on one approach and high on another, green time can be redistributed from the approach with low demand to the one with high demand as necessary. It only when all approaches have high demand that adaptive control is expected to have relatively little benefit compared with other traffic signal control schemes. Table 3.1 and Figure 3.1 outline these relationships.

Demand on Approach/Phase 1	Demand on Approach/Phase 2	Comparative Benefit with Pretimed	Comparative Benefit with Actuated
Low	Low	Optimal	Medial
Low	Medium	Medial	Medial
Low	High	Optimal	Medial
Medium	Low	Medial	Medial
Medium	Medium	Minimal	Optimal
Medium	High	Medial	Medial
High	Low	Optimal	Optimal
High	Medium	Medial	Medial
High	High	Minimal	Minimal

Table 3.1: Comparative Benefits of Adaptive Control vs. other control methods based on demand regimes

Table 3.1 represents the generally *expected* benefits of adaptive control as compared with other types of traffic signal control, and is meant to help provide a justification towards the benefit of research into adaptive control. Experimentation under simulated conditions, which will be conducted, should help to demonstrate the benefits of adaptive control even more clearly. Field conditions may provide variations from these benefits, but, again, should, over a sufficient trial period, demonstrate similar benefits.

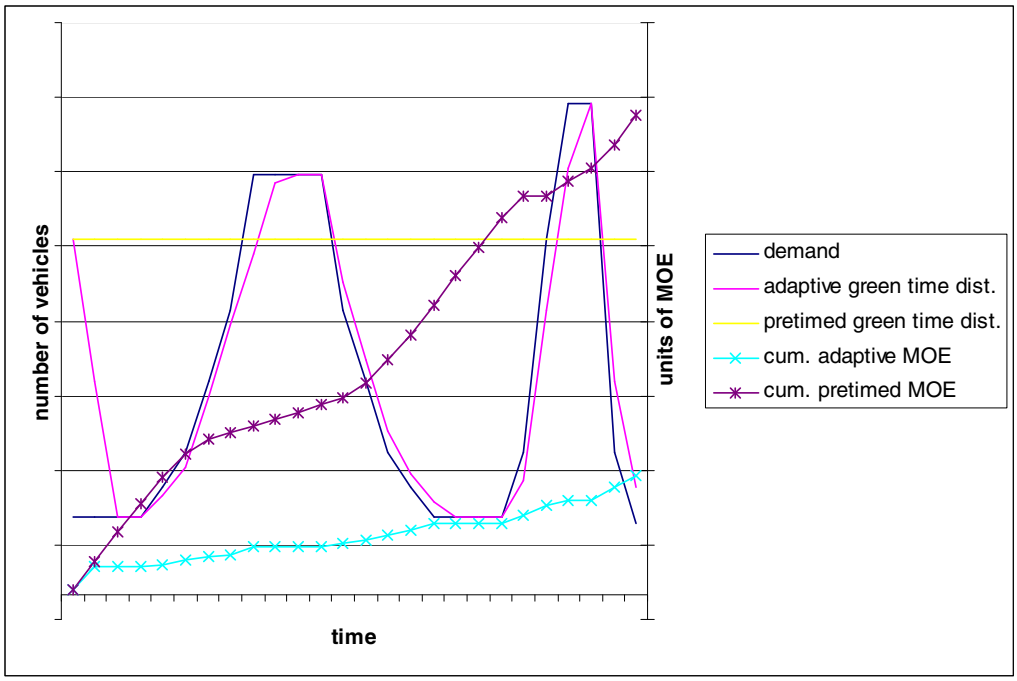

Figure 3.1: Generalized quantification of improvement of measures of effectiveness of Adaptive Control vs. Pretimed Control

The aim of Figure 3.1 is to show the benefit of using adaptive control in a "volatile" demand situation. Figure 3.1 depicts adaptive and pretimed distribution of green time based on traffic demand at a given approach, i.e. green time is provided in terms of demand. It can be seen that the pretimed green time distribution does not change over time. That is to say, the pretimed control will always provide green time for the same demand level. Adaptive control will allow the green time to vary as demand varies. Thus, as time passes, the measure of effectiveness (MOE) to be minimized (delay, number of stops, etc.) will increase much more quickly for pretimed control as for adaptive control.

The difference between the demand satisfied by adaptive control and the actual traffic demand is a function of the ability to predict the actual demand. The slight lag

shown here would be typical of an auto-regressive integrated moving average (ARIMA) type model of prediction used in time-series analysis; other predictive measures may simply try to replicate the shape of the demand curve. This will be discussed further in Chapter 6.

The examples above are rather simplified cases discussing the benefits of adaptive control over other forms in terms of management of MOEs. Real-world cases are much more complex, involving multiple phases, approaches, detection capabilities, etc. It is expected that the benefits will not be as dramatic as the above items depict. However, the potential benefits of the use of adaptive control are clear, and capturing, understanding in the proper context, and incorporating these real-world complexities into the adaptive control procedure, which are among the primary aims of this research, will bring implementation of the procedure much closer to achieving that potential. That implementation begins with looking at previous research into adaptive control, which is done in Chapter 4.

Chapter 4: Review of Previous Research in Adaptive Control

Over the past 20 to 30 years, research into the area of adaptive signal control has gradually been increasing, and practical applications are beginning to be seen around the world,. The purpose of this chapter is twofold; one is to note the major (in terms of extent of application and/or scope of research) adaptive control methodologies and their shortcomings which may be addressed by this research. The other is to qualitatively compare the potential of this research with more recent forays into adaptive control to confirm the validity of this research.

MAJOR METHODOLOGIES

Over the course of performing a literature review, it was found that there are four methodologies that stand out from other attempts at adaptive signal control. They are significant due to their relative acceptance in the field, as well as the relative extent of their real-world implementation.

The Optimized Policies for Adaptive Control (OPAC) methodology is a system first proposed by Nathan Gartner at the University of Massachusetts at Lowell in the early 1980's in a study for the Federal Highway Administration [Gartner (1983), Andrews et al. (1998)]. The Split Cycle Offset Optimization Technique (SCOOT) was also developed in the early 1980's, by the Transport Research Laboratory in the United Kingdom [Greenough and Kelman (1998), Jhaveri et al. (2003)]. The Sydney Coordinated Adaptive Traffic System (SCATS) is somewhat newer, having been created in the early 1990's by the Roads and Traffic Authority of New South Wales, Australia

[Lowrie 2001]. The Real-time Hierarchical, Optimized, Distributed, Effective System (RHODES) is the newest of these four systems, having been produced in the mid-1990's at the University of Arizona at Tucson [Mirchandani and Head (2001)].

SCOOT and SCATS generally use a cycle-based approach on a network, adjusting the cycle times, splits of the cycle and offsets among cycles in the network to optimize an MOE. OPAC and RHODES vary somewhat from this, with OPAC being cycle-based and RHODES being phase-based. Both typically work on the concept of a rolling horizon approach, which optimizes (often using a dynamic approach) an MOE over a fixed prediction horizon and then extends the horizon by a fixed time step and reiterates the optimization until an optimal split of the given cycle is found. Advancements in approaches to OPAC have allowed for some variability in network-wide cycle lengths [Gartner et al. (2001)].

RECENT RESEARCH

Often using the above major methodologies as a basis, recent research into adaptive control has attempted to improve upon these approaches to address what were seen as deficiencies. For instance, three pieces of research revealed attempts to improve SCOOT and SCATS while working under their basic concept of operating on the three parameters of cycle length, phase split and offset. Chiu and Chand (1993) proposed to implement fuzzy logic decision making to individual intersections, rather than the network approach favored by SCOOT and SCATS. Porche and Lafortune (1997) also used a decentralized system, and allowed the use of non-cyclical signal plans at each intersection. Liu et al. (2001) put forward an approach that replicates the SCOOT/SCATS concept, but adds feedback based on on-line delay estimation.

Other approaches did not rely on a specific previous methodology, but proposed improvements to adaptive control in general. The TRYS system, described by Hernandez et al. (1999) suggested adding an additional decision-making layer to allow adaptive control to more easily handle high-demand or other complex traffic scenarios. (The dilemma with these scenarios was touched upon in Chapter 3.) Roozemond and Rogier (2000) propose a similar decision-making mechanism, but with each traffic signal as an independent agent. Tomer et al. (2004) take a rather novel approach, considering the wave formation of traffic flow at a signalized intersection, with the signal causing what they refer to as a localized periodic inhomogeneity; the overall wave formation could then be optimized for flux by controlling the frequency of this inhomogeneity occurs (i.e. controlling the change in signal state).

APPROACH BASED ON PREVIOUS RESEARCH

The previous applications all contain at least one of the following deficiencies:

- Fixed cycle length and/or fixed step for variation of cycle length
- Utilization of demand data only
- Fixed coordination of signals along an arterial or through a network
- Insufficient flexibility of prediction and or optimization parameters

In order to begin to address these deficiencies, the focus has been on two major points of functionality to be incorporated into the proposed adaptive control algorithm. They are the following:

- Independent phase length determination: It is difficult to fix an intersection cycle length that optimizes the performance of an individual intersection. This usually

requires incrementation and analysis of fixed cycle lengths through simulation. The cycle may also be set to its optimum on a network basis, accomplished for pretimed control by such programs as PASSER II [TTI (2003)] and TRANSYT-7F [TRC-UF (1988)], or for adaptive control by the methodologies above. The aim was to create an algorithm that sets a cycle length for each individual intersection based solely on its own traffic streams. In this manner, if coordination or some other network priority is desired, it can be accomplished by the algorithm through optimization of an MOE for the traffic stream on the approach seeking this priority; it was also desired that the algorithm be flexible enough to allow the input of this priority into the optimization process.

- Arrival-based optimization: Nearly all adaptive control methodologies rely solely on the demand observed and/or predicted during a cycle to optimize the timing at an intersection. Herein, incorporation of the arrival times of individual vehicles into the optimization will also be attempted. This will allow the algorithm to perform a kind of actuated function, where vehicles present (or anticipated to arrive) at one approach will be serviced when vehicles are not present/anticipated at another approach, regardless of the demand, which may be aggregated in different portions of the phase or cycle.

In order to allow the algorithm to optimize using the most recent traffic data possible, a feature of OPAC and RHODES will be adapted: the iteration of the algorithm over time steps as traffic data are updated. Other desired features of the algorithm, which were noted in very few of the methodologies, will include:

- The ability to set the analysis horizon (the time period over which optimization will be performed) and the time step (the time between iterations of the optimization).
- The potential to allow the horizon to be optimized (a feature whose importance is reflected in a study performed by Lin and Cooke (1986)).

In developing a new algorithm to address the shortcomings of the previous research described in this chapter, the individual processes will be considered, beginning with detection in Chapter 5.

Chapter 5: Detection in Adaptive Control

In this chapter, two concepts surrounding traffic data detection will be discussed: the function of detection in the adaptive control process and the effect of present and upcoming detection technologies on the effectiveness of the adaptive control process.

FUNCTION OF DETECTION

In the adaptive control process in its current state, field detection serves to obtain the arrival times for vehicles approaching an intersection. These arrival times will be stored in an array classified by their approach and a vehicle ID. This array will be independent of the array that will be used in the adaptive control process. Its purpose is merely to record the actual arrival times measured in the field, which will be used to predict arrival times to be used in the optimization. (The prediction process is discussed in Chapter 6 of this document.) In order to minimize storage requirements, this array will only be held historically as needed by the prediction process.

There are several assumptions made in the collection and use of this data. One is that the time that is placed in the array is the time at which the vehicle actually arrives at the intersection. At the early experimental stages, this arrival time was generated based on common distribution functions such as uniform or Poisson. However, in the field, this arrival time is difficult to measure. In the simplest case, a single detector may be considered. It may have one of two general placements, each of which brings its own issues:

- At the stop line: this allows the measurement of the arrival time at the intersection of vehicles moving through a green phase, but becomes problematic for vehicles arriving during a stop phase, since only a small number of vehicles (possibly as few as one) can sit on a detector and have their arrival times accurately recorded. Vehicles that proceed after a red signal has turned green will not have their desired arrival times (i.e. the time at which they "experience" the signal state) recorded.

- At a distance upstream of the stop line: this allows determination of the arrival time of a vehicle continuously moving at its desired speed at the detector and through the intersection, but may become problematic for vehicles in other situations, generally those that are required to change speeds between the detector and the intersection, such as to stop when encountering a signal state change, to slow to make a turn, to avoid an incident, or to perform a passing maneuver.

The preferable approach of the two is to place the detector upstream of the stop line, as this will allow at least some determination of the arrival time of a vehicle in any situation. The accuracy of the actual arrival time at the intersection based on the time recorded at the detector will be based on how vehicle travel is modeled after passing over the detector. In any case, an upstream detector would require the ability to capture instantaneous speed of the vehicle in order to model this. There are at least two possible approaches:

- Capture the time at the detector and add distance to intersection divided by average speed of all vehicles.

- Capture the time and speed of each vehicle, assign it to some travel regime (based on ranges of speed) and calculate an arrival time based on that regime.

The second approach is more promising in terms of accuracy, but would certainly benefit from the addition of another detector, again with spot-speed capture capability, to help determine what approach situation the vehicle is facing (by examining the change in speed over the two detectors). An approach similar to this will likely be implemented in future incarnations of the adaptive control process.

There is another assumption made herein; this is that all vehicles arrive at a single "intersection point" once their arrival time has been determined. This leads to two characteristics of the traffic stream: all vehicles will arrive downstream of an upstream detector and all vehicles queued at a red signal will be able to depart the intersection before it turns red again. These characteristics do indeed oversimplify the true behavior of a traffic stream, but may prove to be very useful for the demand regimes stated in Chapter 3 as deriving the most benefit from adaptive control: low- to medium-demand traffic streams.

In any case, utilizing a basic scheme for detection involving state-of-the-practice techniques may produce, essentially, imperfect arrival times in some cases. A scheme that mixes several strategies, such as the use of the capture-and-add strategy for low- to medium-demand regimes and the capture-and-assign strategy for high-demand regimes, may prove to be most appropriate. Further research may lead to application of and experimentation with a number of the techniques discussed, focusing on a mixed detection strategy.

DETECTION TECHNOLOGIES

Optimally, consideration of detector technology should involve the following three factors:

- Cost: Utilization of as much of the existing detector equipment as possible should be considered. If the replacement of that equipment is required, it should be low cost in terms of both installation and maintenance.
- Practicality: Consideration should include the ability to place the desired detectors where they are needed, or whether issues such as proximity of intersections in a network, weather conditions or physical obstacles may hinder placement. Also to be considered is the conformity of the detector system in a network, i.e. detector types and configuration should be as uniform as possible throughout a network
- Accuracy: The ability of the detector technology to provide data that reflect actual field conditions is vital. Sensitivity, ability to deal with various conditions (e.g. vehicle configurations, traffic demands, etc.) and deterioration of data accuracy over time are issues to reflect upon here.

The most common detector technology in use in this country is the inductive loop detector (ILD) [Coifman (1999)], which, in simple terms, generates a current as a vehicle passes over it due to the vehicle's magnetic influence on the inductive loop. ILDs can be used as a single placement or in series to detect a variety of vehicle and/or traffic stream characteristics. Although they are susceptible to damage from climatic conditions, passage of heavy vehicles, or any factor that can cause damage to the pavement in which they are embedded, the investigated research generally agrees that they are a relatively reliable and inexpensive detection means.

The functioning of the detectors themselves may be relatively simple, but there have been advances in detector card technology, which involves receiving and interpreting the currents generated by ILDs, and processors, which receive the detector card data and output the required traffic data, as well as the methodologies by which the processors handle the detector card data. These advances have allowed ILDs to meet the needs of many traffic data users, even in single loop placement. Oh and Ritchie (2001) and Oh et al. (2001) use the ability of detector cards to interpret ILD signatures (graphs of the amplitude of the generated current versus their duration) to determine vehicle speeds, classifications and even intersection movements with single loop detectors. Petty et al. (1998) and Coifman (2001) take advantage of improved processor speeds to implement algorithms that allow determination of travel times and speeds from single loop detectors.

As implied earlier, more than one ILD may be necessary for our purposes (the detection and prediction of arrival times), but it may be possible to implement at least the capture-and-assign strategy through a single detector. Multiple detectors may more practically serve the purpose considered by Coifman (1999), to ensure detector reliability by quick error perception.

Advanced detection technology may allow easier implementation of the proposed detection strategies. This is especially true in the case of multi-lane urban network applications, where incidents, erratic driver behavior and transit use make the ability to identify specific vehicles important to gathering data. Video detection is such a technology; Panda and Anderson (1999), representatives of the manufacturers of Autoscope, a common video detection system, tout its reliability, competitive cost with ILDs in terms of installation (through simplified configuration), setup (through remote

management) and maintenance, and use in a number of traffic data requirements, including adaptive control implementations such as SCOOT and SCATS. Video detection does have some drawbacks; physical placement may be difficult because of availability of mounting structures, blocking by obstacles and angles required for proper detection, and they may also not operate as effectively at night or during obscuring weather conditions such as rain, snow, fog and dust.

Video detection, despite its general reliability in comparison with the ILD, may not be accurate enough to perform the vehicle identification that may be required to achieve optimum detection accuracy for the proposed adaptive control system. For this, it may be desirable to consider advanced technologies such as those surveyed in Europe by Van Arem et al. (1997); these included image processing and/or automatic license plate detection in conjunction with video detectors, and smart cards or other types of electronic tags within vehicles. Although these technologies are certainly not common now, nor are they anticipated to be so in the near future, their use on even some fraction of vehicles on intersection approaches may yield significant improvements in detection.

In general, implementation of the above techniques and technologies are apt to create an enhanced detection process, which has direct bearing on the ability to predict traffic data, a process described in Chapter 6.

Chapter 6: Prediction in Adaptive Control

Before discussing traffic data use in the proposed adaptive control process, a caveat must be considered that is applicable to all traffic data use, but particularly key here: the entire process will be no more reliable than the data that it is provided. In essence, the focus in this chapter is to attempt to provide the adaptive process with data that most accurately represent actual conditions.

Most traffic control schemes operate based on previously measured and/or calculated traffic data. Even actuated control schemes rely on pre-existing traffic data for many of their parameters. In truth, adaptive traffic signal control suffers from this limitation as well; it is not claimed to be a "real-time" control scheme. What adaptive control does better than nearly any other scheme, however, is to utilize what is often called short-term traffic prediction or short-term traffic forecasting.

Short-term traffic prediction is the attempt to deduce characteristics of a traffic stream for some given future time horizon. The ways in which this is done are myriad, but they essentially boil down to consideration of three basic elements, which were touched upon in Chapter 2. These are:

- Length of time horizon used
- Use of multiple time horizons
- Prediction process

These will not be discussed individually herein; rather, prediction will be examined in terms of these three elements under the following headings:

- Previous research into the area of short-term traffic prediction, and
- Selection of one or more traffic prediction processes for this research

RESEARCH INTO SHORT-TERM TRAFFIC PREDICTION

The key piece of traffic data to be used in the proposed adaptive control process is the vehicle's arrival time to an approach, or the sum of the arrival time of the previous vehicle at the approach and the interarrival time between the two vehicles. For constant demand, a Poisson process has often been used to calculate this interarrival time, with the parameter being the mean arrival rate (Pignataro (1973) and May (1990)). In fact, this is the process being used currently in the proposed process.

However, for a more dynamic traffic demand scenario, it is unlikely that the constant-parameter Poisson process will be appropriate to predict interarrival times between vehicles. Recent research has taken this into account and posited prediction methodologies that are more responsive to this dynamic traffic demand. Much of it has been centered on the use of time-series analysis to predict certain characteristics of individual vehicles and/or the traffic stream in general. Time-series analysis essentially allows data previously measured to be incorporated into the prediction of future data. One of the models used to carry out time-series analysis is the ARIMA (Auto-Regressive Integrated Moving Average) model, proposed by Box et al. (1994). Through its auto-regressive aspect, it accounts for patterns of correlation that arise in the data, and through its moving average aspect, it accounts for increasing or decreasing trends in the data. They take the general form of:

$$\Phi(B)\, Z_t = \Theta(B)\, a_t$$

where B represents the number of time steps (or, in our case, the number of vehicles) backwards on which to operate, $\Phi(B)$ is the auto-regressive operator, $\Theta(B)$ is the moving average operator, Z_t is the observation at time t (or, in the present case, the interarrival

time for vehicle t), and a_t represents any random input arising at vehicle t. If there is no random input, then both $\Phi(B)$ and $\Theta(B)$ can operate on the previous observations to produce future observations.

There are many permutations on the general ARIMA formulation that allow it to conform to expectations concerning the previous or future data, such as differencing, filters and feedback operators. Ishak and Al-Deek (2002), for instance, outline a time-series formulation that predicts speed over a short-term horizon using the auto-regressive operator in conjunction with a parameter that uses a form of the moving average operator in a two-step procedure. They report that "relatively accurate speed predictions" were made using this form. Another formulation, by Hamed et al. (1995), predicts traffic volumes over one-minute intervals using only the moving average operator with differencing, with what could be considered to be accurate results, as well. Hobeika and Kim (1994) take an approach based on general time-series analysis using, instead of an auto-regressive component, an operator based on traffic data measured upstream of the point of interest, with results they found to be improved from using a moving average component alone.

Van Arem et al. (1997) discuss a number of other advances in short-term traffic prediction of speed and/or demand. These include the use of neural networks, which can more easily recognize patterns in the traffic data, as well as such other concepts as cluster analysis.

Of course, none of the above approaches address the issue of prediction of interarrival times for vehicles, nor does most other research into short-term traffic prediction. They do, however, speak to the viability of forecasting traffic data

characteristics into the very near future (no more than 15 minutes, and as near as one minute). Enns and Li (2004) examine time-series prediction of interarrival times from an operations approach (the optimum lot-sizing problem), with relatively low coefficients of variation. This kind of scenario may be considered analogous to vehicle processing at an intersection for our purposes, and, in combination with the traffic-related research examined herein, can provide insight into prediction methodology for the proposed adaptive control process.

POSSIBLE PREDICTION METHODS FOR THE PRESENT RESEARCH

As previously stated, in its present form, the proposed adaptive control process uses a process that generates interarrival times based on the Poisson process, an approach that has allowed experimentation using the proposed process but which can prove inaccurate for fluctuating traffic demand and/or high-demand regimes. It is not, however, the aim of this research to conduct an in-depth investigation into prediction methodology for adaptive signal control. Rather, it is stipulated that there exist methodologies that may allow our process to function optimally, i.e. make full use of its arrival-based optimization technique (see Chapter 7 for details on the objective function). It is noteworthy that, even in the absence of an "accurate enough" solution, experimentation shows that the proposed process is still demand-responsive (see Chapter 9 for details on experimentation).

Given the dynamic and, even for a constant demand regime, possibly sporadic nature of interarrival times in a traffic stream, it is improbable that either basic linear regression or fitting of standard probability functions will allow the prediction of these times with any great degree of accuracy. Based on the above cited research and common

techniques practiced for the calculation of interarrival times, the following can be consider among several possibilities for interarrival time prediction for the proposed adaptive control process:

- Direct time-series prediction of interarrival times: This process would use detected differences between arrival times of vehicles in a time-series formulation to determine the time until the next vehicle would arrive. It is very unlikely that the B value for each portion of a general formulation (i.e. the parameter of each portion) would remain constant, given the dynamic nature of traffic streams in general, and ones to be impacted with adaptive control in particular. For this reason, it would likely be fruitful to impose some kind of feedback mechanism that causes the parameters of the functions to change based on the incoming detector data; this type of mechanism was considered by Van Arem et al. (1997). Further experimentation may also lead to incorporation of filters or other components to the formulation as necessary, although the goal is to keep the formulation as simple as possible to minimize computation time.

- Poisson prediction with variable-demand mean: This process would use a time-series methodology such as the one proposed by Hamed et al. (1995) to predict demand, and then use that demand to calculate the mean interarrival time (the time divided by the number of vehicles arriving during that time) to be used as the Poisson process parameter. In this manner, the parameter would vary with demand, allowing the preservation of at least the general arrival patterns of vehicles over time. This is not as likely to capture the sporadic nature of the interarrival times, but is probably much easier to apply and may prove to be more

practical, as it is more practical to measure, calculate and predict demand than interarrival times.

- Incorporation of upstream traffic data into prediction: This would use data detected upstream of an intersection to predict arrivals of vehicles downstream, whether directly, through prediction of travel times and/or speed for future vehicle arrivals, or through incorporation into a time-series formulation as feedback, much as proposed by Hobeika and Kim (1994). This is likely only feasible for isolated rural intersections, as intersections in an urban network would likely have a high degree of upstream interference (other intersections, turning vehicles and mid-block sources/sinks) that would hinder the usefulness of that data.

Further research might involve experimenting with all three of these methods to determine which one would be most favorable to integrate into the proposed adaptive control process. As indicated, accuracy would not be the only factor in considering these methods; examination of ease of implementation, required computation time, and detection capabilities would also be necessary. Consideration, within any prediction process, of such factors as the forecasting and historical horizons and step times to be used is likewise important, since, as Ishak and Al-Deek (2002) demonstrate, they are significant to the accuracy of predicted values.

Furthermore, future research into the prediction process itself may result in the consideration of methodologies where the overriding factor is the accuracy of prediction. As Van Arem et al. (1997) imply, this will likely be a type of methodology that can incorporate a high level of pattern recognition, such as neural networks or ARIMA models with a combination of filtering and feedback. This would take advantage of the

prevalence of arrival patterns in a traffic stream, particularly in urban networks, where proximity of signalized intersection, time-of-day patterns and incidents can be highly influential on interarrival times.

Addressing the accuracy of the prediction process is paramount in the performance of the optimization to be considered in Chapter 7; without this accuracy, the aims of the adaptive control process are much less likely to be realized.

Chapter 7: Optimization in Adaptive Control

In this chapter, use of traffic data derived from the previous steps in the process to implement a traffic signal control scheme that optimizes a measure of effectiveness is examined. To do this, the concept of a measure of effectiveness in terms of traffic signal control is first explored. Then, formulation of an objective function that operates based on measures of effectiveness is investigated. Finally, the optimization solution search is considered.

SELECTION OF A MEASURE OF EFFECTIVENESS

In terms of traffic control, a measure of effectiveness (MOE) serves two purposes:

- To provide a numerical basis over which an algorithm will perform an optimization
- To provide an indication of the performance of any control process (i.e. an upper or lower bound of the measure of effectiveness will indicate free flow of traffic, and the control process will attempt to reach this bound)

There are a number of MOEs used in traffic control. In choosing an MOE, one must consider:

- The ability to collect data on, calculate, and work with the measure
- The ability of the measure to reflect the true performance of the control process

A useful and well-accepted consideration of the value of several MOEs is made in Herman's two-fluid model of traffic flow [Herman and Prigonine (1979)]. Herein, travel time on a link or set of links is acknowledged to be "the most reliable single variable in the traffic assignment process", and thus probably also the best MOE to describe

intersection performance. However, vehicle travel time is difficult to measure in the field for a signalized network, with it normally being collected by sending pilot vehicles into the network. (This method of measurement was used for adaptive signal control in Andrews et al. (1998).)

Using this MOE as a baseline, it was also determined (theoretically and experimentally) through the two-fluid model that there is essentially a linear relationship between travel time and stopped delay. However, there is a much more tenuous connection between travel time and number of stops, relating generally to a parameter based on "the quality of the traffic network". Other MOEs are discussed, such as average flow and speed, and concentrations of moving and stopped vehicles, but these are not as useful in the realm of adaptive signal control, as they require measurement over longer periods of time.

In effect, stopped delay can be judged as perhaps the most useful MOE for the adaptive control process proposed here. This is borne out by its use as the major MOE in other adaptive control methodologies [Gartner (1983); Jhaveri et al. (2003); Lowrie (2001); Mirchandani and Head (2001)]. (Some methodologies such as OPAC have also incorporated the number of stops as a weighted optimization variable to prevent this MOE from exceeding some upper bound [Gartner (2001)]; it is likely that the proposed adaptive process will tend to carry out that objective without imposing it as a constraint, although the possibility of using this constraint in future incarnations of the proposed process will be explored in further research.)

A relatively accurate measurement of stopped delay can be made using detectors at each approach and a clock tracking the traffic signal status. How this measurement will

be made and incorporated into the adaptive control algorithm will be seen in the formulation of the optimization model.

FORMULATING THE OBJECTIVE FUNCTION

To begin consideration of an objective function involving stopped delay for use in the adaptive control algorithm, a simple traffic-related problem is examined, where there are three approaches to an "intersection" that has only one direction leading away. Such a configuration is depicted in Figure 7.1. Let i = 1,…,N be the set of N approaches (in this case, three) to the intersection. As a vehicle arrives at an approach, it is assigned a vehicle ID, j. Therefore, the time that a vehicle arrives at the intersection from an approach can be labeled as t_{ij}.

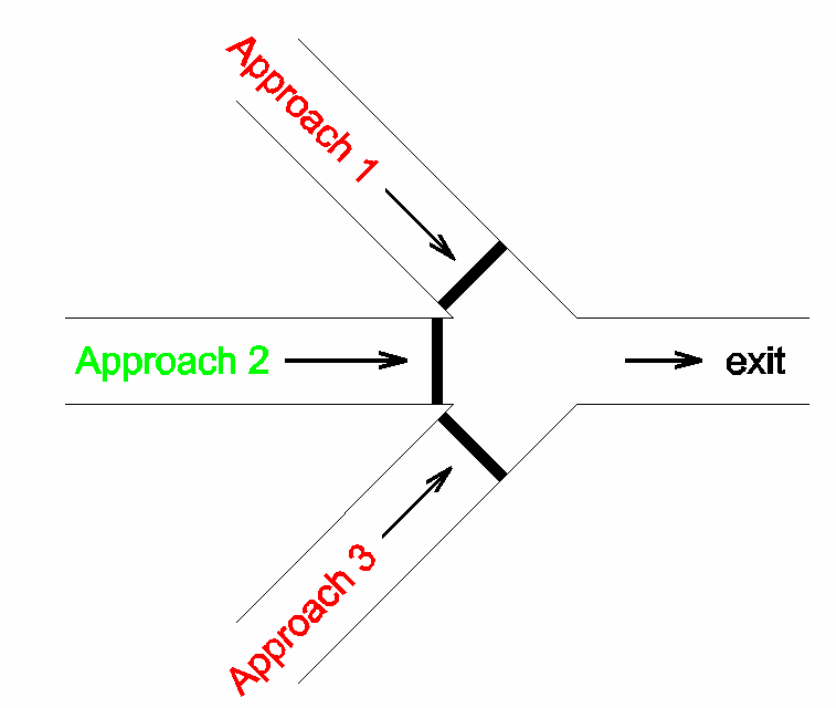

Figure 7.1: Simple two-phase intersection.

Each vehicle arriving from an approach wishes to utilize the intersection to travel in the one direction leading away from it, but only one approach at a time will be granted a green light to allow this motion, while the others have a red light. After each approach receives a green light once, a cycle will be considered to have been completed. The approaches can each receive a green light in any order over the course of the cycle; let k = 1,…,M be the set of green phases, a phase being the state of an approach experiencing a green light. Thus, each approach would have a k index, k_i, independent of the value of its i index, e.g. approach 1 can be the third one to receive a green phase during a cycle. Currently, each k index is considered fixed to each particular i index (e.g. approach 1 will always be third in the cycle).

H can be considered to be the horizon (period of time) over which to examine the arrival of vehicles to the intersection. (This will typically be on the order of the desired length of the cycle.) At this point, Figure 7.2 can be considered, where the arrival of vehicles over a horizon H is examined.

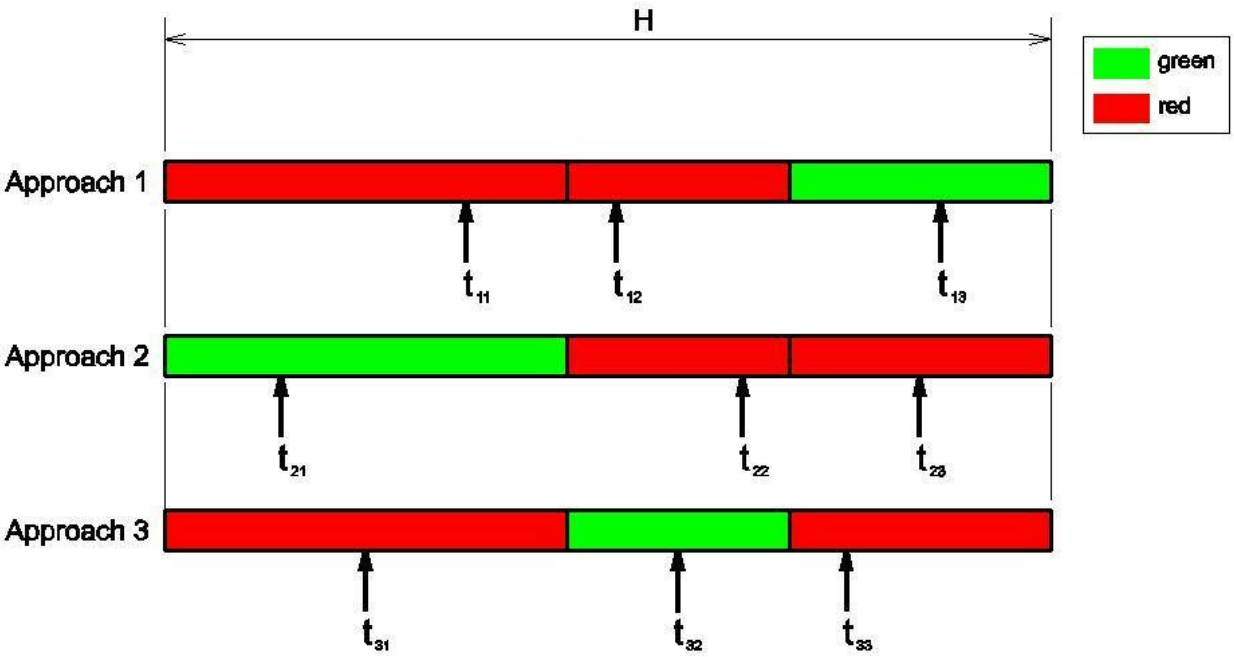

Figure 7.2: Preliminary evaluation of approach arrivals at an intersection over H.

In considering stopped delay, there are three possible conditions for which a vehicle can arrive at any given approach. These are:

- A vehicle can arrive while the approach does not have a green phase, but before the time the green phase begins.
- A vehicle can arrive while the approach has a green phase.
- A vehicle can arrive while the approach does not have a green phase, and after the time the green phase has ended.

If a vehicle arrives while the approach does not have a green phase in either condition, it must experience stopped delay, while it will experience no stopped delay if it arrives during the approach's green phase. The manner in which the stopped delay for

each vehicle that experiences it is calculated can be considered with the definition of more values related to the cycle:

- cs (for cycle start): the time point at which the cycle begins
- ce (for cycle end): the time point at which the cycle ends, which is cs + H
- λ_k: the proportion of the horizon that has passed at the end of phase k. Thus, a phase would end at time cs + λ_kH. (Note that, in this instance, ce also comes at time cs + λ_3H.)

Now, the calculation of stopped delays, as depicted in Figure 7.3, can be addressed.

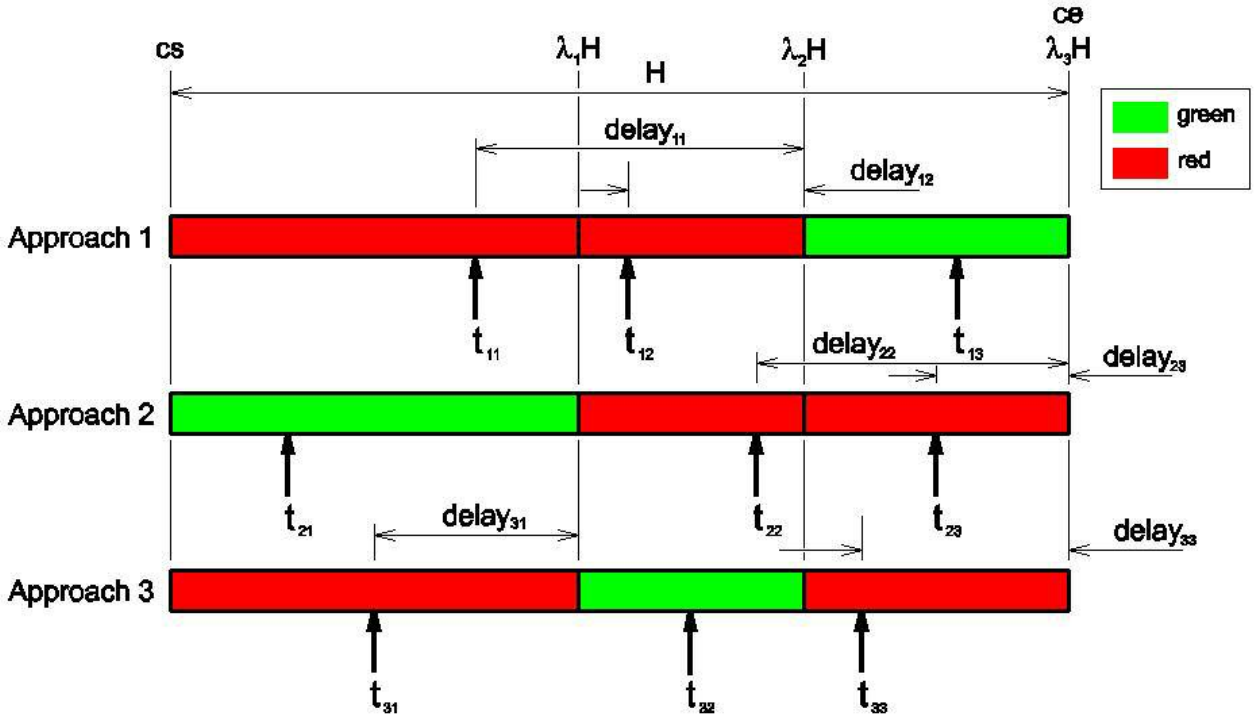

Figure 7.3: Evaluation of approaches to an intersection over H, with cycle values.

The stopped delays can be calculated as:

- $delay_{11} = (cs + \lambda_2 H) - t_{11}$
- $delay_{12} = (cs + \lambda_2 H) - t_{12}$
- $delay_{13} = 0$

- $delay_{21} = 0$
- $delay_{22} = (cs + H) - t_{22}$
- $delay_{23} = (cs + H) - t_{23}$
- $delay_{31} = (cs + \lambda_1 H) - t_{31}$
- $delay_{32} = 0$
- $delay_{33} = (cs + H) - t_{33}$

The calculations for each arrival state (red before green, green and red after green)) can be generalized as follows:

- red before green: delay = $(cs + \lambda_{(ki-1)}H) - t_{ij}$
- green: delay = 0
- red after green: delay = $(cs + H) - t_{ij}$

Each approach will experience each of these states once during a cycle. Summing the delay of all vehicles arriving at an approach during a cycle based on each vehicle's arrival state will yield the total stopped delay for the approach during that cycle.

The significant observation to be made from Figure 7.3, once calculation of stopped delay can be derived, is that the value, or even the existence, of stopped delay, can be altered based on variation of the λ_k values. For instance, the value of λ_1 can be reduced to attempt to reduce, or eliminate, $delay_{31}$, but the creation of $delay_{21}$ is possible if λ_1 is reduced too greatly. To attempt to capture the vehicle arriving at t_{22} in approach 2's green phase the value of λ_1 can be increased, but a $delay_{32}$ may be created in the process. These λ_k values become the decision variables in the optimization, which will be selected to minimize the dependent delay variables.

The λ_k values are limited in their variation by only two constraints: they are between zero and one and each subsequent λ_k value is greater than the one preceding it. These constraints are not limiting enough to prevent any vehicle from arriving at the intersection during any arrival state; therefore, calculation of stopped delay in any arrival state for any arriving vehicle must be anticipated.

In essence, both possible stopped delays will actually be calculated for every vehicle each time the objective function is carried out. Then each calculated delay will be excluded or included in the summation based on the judicious use of the following "dummy variables":

- x_{ij} equals: 1 if $t_{ij} - (cs + \lambda_{(ki-1)}H) \leq 0$ (i.e. the vehicle arrives before the beginning of its approach's green phase)
  0 otherwise

- y_{ij} equals: 1 if $(cs + \lambda_{(ki-1)}H) < t_{ij} \leq (cs + \lambda_{ki}H)$ (i.e. the vehicle arrives during its approach's green phase)
  0 otherwise

Both possible stopped delays for a vehicle arriving at t_{ij} involve the subtraction $ps - t_{ij}$, and will include the addition of $\lambda_{(ki-1)}H$ if $x_{ij} = 1$ and the addition of H if $x_{ij} = 0$. If $y_{ij} = 1$, the vehicle should experience no stopped delay. Thus $\lambda_{(ki-1)}H$ should be multiplied by x_{ij}, H multiplied by $(1 - x_{ij})$, and the entire delay calculation multiplied by $(1 - y_{ij})$ to produce the delay formulation for a vehicle arriving at t_{ij}. It can be seen in Table 7.1 how this results in the proper delay calculations for the sample set of vehicle arrivals:

Delay ID	x_{ij}	k_i	before green delay	$(1 - x_{ij})$	after green delay	y_{ij}	$(1 - y_{ij})$	Stopped delay
$delay_{11}$	1	3	$ps + \lambda_2 H - t_{11}$	0	$ps + H - t_{11}$	0	1	$ps + \lambda_2 H - t_{11}$
$delay_{12}$	1	3	$ps + \lambda_2 H - t_{12}$	0	$ps + H - t_{12}$	0	1	$ps + \lambda_2 H - t_{12}$
$delay_{13}$	0	3	$ps + \lambda_2 H - t_{13}$	1	$ps + H - t_{13}$	1	0	0
$delay_{21}$	0	1	$ps + \lambda_0 H - t_{21}$	1	$ps + H - t_{21}$	1	0	0
$delay_{22}$	0	1	$ps + \lambda_0 H - t_{22}$	1	$ps + H - t_{22}$	0	1	$ps + H - t_{22}$
$delay_{23}$	0	1	$ps + \lambda_0 H - t_{23}$	1	$ps + H - t_{23}$	0	1	$ps + H - t_{23}$
$delay_{31}$	1	2	$ps + \lambda_1 H - t_{31}$	0	$ps + H - t_{31}$	0	1	$ps + \lambda_1 H - t_{31}$
$delay_{32}$	0	2	$ps + \lambda_1 H - t_{32}$	1	$ps + H - t_{32}$	1	0	0
$delay_{33}$	0	2	$ps + \lambda_1 H - t_{33}$	1	$ps + H - t_{33}$	0	1	$ps + H - t_{33}$

Table 7.1: Delay formulation based on inclusion of dummy variables

If the delays in the last column are summed, the total delay, the value to be minimized by varying the λ_k values for the cycle, is obtained. (Note here that λ_0 should always be intitalized to zero.)

Another delay element that can be incorporated into the optimization is queue startup time. This is the delay experienced by vehicles as they begin to depart the intersection after the signal on their approach turns green. This delay was quantified by Greenshields et al. (1947). Based on Greenshields' values, the following delay regime for queue startup has been selected:

- 1st vehicle in queue: 2 seconds
- 2nd vehicle in queue: 3 seconds
- Each vehicle in queue after 2nd vehicle: 4 seconds

The total delay caused by queue startup experienced for each green phase is clearly highly dependent on the number of vehicles queued at an approach. This delay can be influential on the λ_k values by causing them to tend towards preventing a vehicle from experiencing a "red before green" state. (Vehicles experiencing a "red after green"

state will have their startup delay counted as the signal cycles back to put them in the "red before green" position.)

In order to model the queue startup delay, the dummy variable x_{ij} is considered; it takes a value of 1 if the vehicle experiences a "red before green" state. If the vehicle arriving at t_{ij} is the last to experience the "red before green" state, all vehicles with j greater than this will have an x_{ij} value of zero, and all those with j equal to or less than this will have an x_{ij} value of 1. Thus, the following expression can provide the startup time for each vehicle based on the Greenshields method:

$$\text{qst} = (2x_{ij} + x_{i(j+1)} + x_{i(j+2)})$$

There is another delay issue to consider: a concept that might be referred to as carryover delay. This is delay that is accumulated by vehicles whose arrival times will not meet the constraints allowing them to be considered in ensuing phases, but have not yet been granted a green phase. In Figure 7.3, the vehicle arriving at t_{11} would fall into this category. Its delay is calculated in the first iteration of the algorithm, but in the next iteration, the cycle start will be considered from cs + λ_1H, removing t_{11} from the delay calculations. (See Chapter 8 for more on the reinitialization process for recursive operation of the algorithm.) In fact, the delay for this vehicle still accumulates until it is removed by the granting of a green phase to its approach, and it will tend to influence the λ_k for its approach.

The procedure for dealing with this carryover delay herein is through accumulation counters in an array based on the number of phases ahead of the approach's green phase. These counters allow proper calculation of the delay for queued vehicles no longer meeting the constraint for the objective function based on the λ_k values for their

phases that change during the ensuing iterations of the algorithm. The function governing this carryover delay and its influence on the λ_k values of the solution is:

$$carryover = \sum_k \left[(counter_k \times (cs + \lambda_{k+1} H)) - tsum_k \right]$$

where $tsum_k$ is the sum of the arrival times about to leave consideration for the delay objective function. The counter for phase is zeroed out when the ensuing horizon begins with phase k being green.

With all of the above considered, an objective function can be written for stopped delay (including queue startup time) as follows:

$$\min_{i,j} delay = \sum_i \sum_j (1 - y_{ij}) \begin{bmatrix} (cs + x_{ij}(\lambda_{k_i - 1} H) + (1 - x_{ij})H - t_{ij}) \\ + (2x_{ij} + x_{i(j+1)} + x_{i(j+2)}) \end{bmatrix}$$

$$+ \sum_k \left[(counter_k \times (cs + \lambda_{k+1} H)) - tsum_k \right]$$

Subject to $0 < t_{ij} - cs \le H$

$$0 < \lambda_k \le 1$$

$$\lambda_{k-1} \le \lambda_k$$

$$x_{ij}, y_{ij} = 0,1$$

In using the above objective function, a set of λ_k values is selected and the optimization is carried out over all vehicles arriving on an approach during the horizon; this is then performed over all approaches to a given intersection to obtain the total stopped intersection delay. The λ_k values can then be varied until a set is found that provides the minimum total stopped intersection delay.

It should be noted that the minimum total stopped intersection delay yielded by the above optimization model is not in fact the actual stopped delay experienced by the vehicles arriving at the intersection. It is, rather, the potential delay that would be experienced if the full cycle were carried out using the λ_k values in the solution set. The experienced delay for the phase itself is the sum of the stopped delay experienced by vehicles about to receive the green in the ensuing iteration, the carryover delay of vehicles arriving before the cycle start time but not yet receiving green, and the queue startup time for those vehicles. It is, indeed, the ability of the algorithm to recalculate the phasing and reduce stopped delay for vehicles that have not yet received green that qualifies it as a phase-by-phase optimization that has the potential to outperform cycle-based adaptive traffic signal control methodologies.

Although the above objective function has been prepared using a basic 3-entrance, 1-exit gating-type configuration, the function should be applicable to nearly any intersection configuration with any number of phases distributed among the various movements. Its applicability will be demonstrated through the experimentation described in Chapter 9.

SEARCHING FOR OPTIMUM λ_K VALUES

The above objective function can be viewed purely as a function of the λ_k values, as these determine the x_{ij} and y_{ij} values. As a function of the λ_k values, the objective function is nonlinear and discontinuous over each phase. So, the application of linear or classical nonlinear programming techniques is not possible. Instead, several other approaches may be viable, including:

- Exhaustive enumeration of λ_k values

- Numerical search procedures
- Metaheuristic search procedures

The choice among these approaches depends on several factors, including how the problem is defined. The relative merits of each approach will be discussed below.

Consideration of Non-Metaheuristic Approaches

At first glance, the optimization seems to have an infinite number of possible λ_k solutions, and thus a numerical search procedure would lend itself ably to the task. Finding the optimum λ_k values would not be guaranteed, but given enough operating time and a suitable method, a solution within some acceptable margin of error of the optimum could be reached. Numerical search methods over non-convex intervals generally involve using a one-dimensional search (e.g. steepest descent, golden section, etc.) of the range over unimodal intervals. The specifics of these types of methods will not be discussed herein, but are readily referenced through literature on optimization; Fletcher (1987) is a good reference for the specifics of these methods. There are drawbacks, however, to the use of these in that, depending on the acceptable error and the initial solution used, they can be time-consuming and may prevent reaching an acceptable solution in a practical time frame. Other limiting factors in using this type of approach are the time step over which the optimization is iterated, as discussed in Chapter 8, and the number of phases over which the system needs to be optimized, which can expand the computation time exponentially.

In light of this, one might consider that there may not be an infinite number of possible λ_k solutions, but that this number is practically limited. It is simplest to consider that the λ_k values, being factors that partition the horizon, should have a limiting resolution based on the system performance. One end of the spectrum of resolutions

might be, for instance, the transmission speed between the processor and the signal controller, which may be on the order of 10^{-3} seconds or lower [AASHTO et al. (2004)], and the other might be the minimum headway between vehicles at freeflow speed, which is generally assumed, based on Greenshields et al. (1947) to be approximately 2 seconds. This puts the number of practically possible λ_k solutions in the range of less than 10^2 to 10^4 per phase. With this lower number of possibilities, it becomes reasonable to consider exhaustive enumeration within the time step of the iterations. A potential drawback to this, however, is that, if the resolution is not small enough, it may not be possible to capture the time difference between arrivals of vehicles at multiple approaches, eliminating the ability to utilize λ_k solutions that may provide delay improvement. (This is, in fact, an additional drawback to numerical search methods, which can exhibit this fault if the acceptable error in the λ_k solution is not small enough.) Another drawback is, like the numerical search methods, as the number of phases to be considered increases, so does the computation time, which must not exceed the iteration time step.

Potential Benefits of a Metaheuristic Approach

If one considers the manner in which delay is calculated, based upon vehicle arrivals, one may actually see, through examination of Figure 7.3, that delay can be optimized most readily by making switches in the signal states only within some interval of these critical points along the horizon. This is to say that delay can only approach a minimum by switching the signal state as quickly as possible after a vehicle passes through an approach that is presently served by a green signal. When considering this, the problem becomes more like a combinatorial optimization problem. In it, the λ_k value for each phase determines the placement of a vehicle into one of three sets: the red before green set, the green set and the red after green set. The delay for each vehicle is a set

value for each of the now-limited λ_k values to be considered; the determination then becomes which λ_k values will allow which delay values to be considered in which set, determining whether the delay value in question will contribute to the overall delay.

It is possible, with the number of possible λ_k solutions reduced to, essentially, the number of vehicles arriving at the intersection during the horizon, to perform an exhaustive enumeration of the potential λ_k combinations to find the optimal solution. It may also be possible to construct a numerical search procedure through an integer programming approach after this reduction. The overall number of combinations using one of these approaches would be on the order of the maximum number of vehicles wishing to utilize a phase during a given horizon to the power of the number of phases. Clearly, though, as the number of vehicles and the number of phases increase, the computation time for an exhaustive enumeration would increase exponentially. For instance, on a 4-phase intersection where a maximum of 30 vehicles need to be processed for a phase, the number of possible λ_k combinations is 30^4, or approximately 810,000 combinations. This type of problem could be easily encountered where a phase processes more than one approach (e.g. one phase for both directions of a street).

A situation where there is a high demand regime to be processed by one phase and lower demand to be processed by the others would create many combinations where λ_k values of zero hold the places in the λ_k array where vehicle arrival times in those phases are nonexistent. Possibilities to streamline an exhaustive enumeration would be the creation of an alternative representation of the λ_k array (e.g. forward or backward star) to eliminate the zero placeholders and reduce the number of combinations. However, without excluding this as a possibility for further exploration, especially for less complex applications, the benefits of such an approach would only be significant in a highly

disparate demand situation; this is not guaranteed to be present, from phase to phase, even for the same intersection. In any demand distribution case, the increase in complexity and overall vehicular demands on intersections to be addressed by the proposed methodology would likely outstrip the benefits of such an approach.

Furthermore, this increase is only considered on a single-intersection basis; operation on an arterial or network basis can render exhaustive enumeration quite cumbersome. A major constraint, the availability of computation time based on the time step of the iterations and the "real-time" demands of the algorithm's operation, leads to the consideration of other time-saving alternatives.

Application of the Metaheuristic Approach

Tabu search is a metaheuristic search procedure that addresses many of the pertinent issues while providing a clear logic to the transition between possible λ_k combinations to be considered. A discussion of tabu search can be found in Glover (1989), but the process can be summarized as follows:

1. An initial solution is generated and its objective value is taken as the current best value.
2. A neighbor (or a candidate, from a list more restrictive than the neighborhood) to the initial solution is selected based on a move.
3. Its objective value is determined:
    a. If the neighbor's objective value is less than that of the initial solution, then that value becomes the current best value.
    b. If the neighbor's objective value is greater than that of the initial solution, then some attribute of the move becomes a tabu restriction of any move

(i.e. a move to a neighbor involving that attribute cannot be made) for some number of iterations (the tabu tenure).

4. If 3a is true, then a non-tabu move is made from the initial solution; if 3b is true, a non-tabu move is made from the neighbor.
5. Non-tabu moves can be made until:
    a. the neighborhood or candidate list is exhausted.
    b. an aspiration criterion (one that allows a move that is considered tabu to be made) is applied.
    c. a stopping criterion (e.g. end of available computation time, minimal improvement in solution, etc.) is met.

Tabu search has been applied to a wide variety of operations research problems, and has had a number of enhancements made to it which have allowed it to tackle rather complex problems in this area. A rather simple tabu search application was used as a framework for the adaptive traffic signal control optimization in its current setup; this is similar to that of Laguna et al. (1990) for the single machine scheduling problem, which essentially involves the basic process described above. Its coding has been included in Appendix 1.

The search can start with the array of λ_k values, which has a number of rows equal to the maximum number of vehicles arriving at a phase and a number of columns equal to the number of phases to be considered. A λ_k is selected for each k, or phase, in the array. Then, a neighborhood of that solution could have all the λ_k values held except for one, which could vary based on a move up and down a particular column. Such a move would be defined as a low influence move. The neighborhood could also allow the columns to switch for that set of λ_k values, so that the green times could be allowed to occur in a

different order during the horizon. This would be defined as a large influence move. The current implementation only uses low influence moves, adhering to the constraint of increasing sequential λ_k values.

In the current case, the set of proportional heuristic solutions is generated (as described below) and chosen from for the initial solution. Based on the occurrence of a non-decreasing delay, a set of λ_k values has one of its elements assigned as tabu. The element and its location in the λ_k set are placed in arrays where they remain for the length of the tabu tenure. If a λ_k set is then selected with the tabu element in the stored location before the tabu tenure is completed, that λ_k set is rejected and another λ_k set is leaped to in the proportional solution space, which is the neighborhood in this particular application. The tabu array can hold as many element-location pairs as the value of the tabu tenure.

The aspiration criteria have a large effect on the ability to explore solutions, since it is quite possible that a tabu or non-neighborhood move may lead to a significantly improved solution compared to the current best solution. The two main applicable aspiration criteria are these:

- an “aspiration by default” criterion, where, if all moves are tabu, a move will be allowed that led to the objective value closest to the current best value
- an “aspiration by influence” criterion, where, if a “high influence” move has been performed since an attribute became tabu, “low influence” moves involving that attribute are now allowed

A version of the “aspiration by default” criterion is used herein, whereby, if all moves are tabu, the selection that was originally next in the proportional solution space is selected.

The "Proportional Heuristic"

The initial solution is also key to exploration of the solution space, in that a good initial solution may significantly reduce the computation time needed to reach a high quality solution. In this case, a solution will be used based on what can be considered a "proportional heuristic". The concept requires starting with a "proportional lambda" for each phase, calculated using the proportion of vehicles wishing to utilize that phase during a horizon to the total number of vehicles arriving at the intersection during that horizon, much like Webster's method for resolving phase lengths in a cycle (Pignataro (1973) and May (1990)). Using this set of proportional lambdas, an initial solution is obtained with the λ_k value for each phase that has the smallest deviation form the proportional lambda for that phase. Mathematically, this initial solution is:

$$\lambda_k = \lambda_{kj} \text{ such that } \min[\lambda_{kj} - \lambda_{(ki-1)} - (\text{\# of veh. for k})/(\text{total \# of veh.})]$$

This initial solution can be used to continue the metaheuristic search, or the proportional heuristic can be used to optimize the λ_k combinations, if desired. To do this, the potential combinations of λ_k values must be exhaustively enumerated, and then ranked for sequential use by the algorithm. This ranking is done by ascending order of the sum of the differences between the proportional lambda and the λ_k value in the combination for each phase. Therefore, solutions which deviate less from the proportional lambda set will be ranked higher and available for use in the algorithm at earlier iterations. Using the solutions based on this ranking system, the algorithm should be able to arrive at a good, if not optimal, solution, much more quickly than ordinal exhaustive enumeration. It should be noted that this situation is not guaranteed, as peculiarities in the arrival time set (e.g clustering arrivals at the extremes of the horizon) may produce an optimum λ_k far from the proportional lambda for a particular phase.

Furthermore, use of the proportional heuristic is only preferable on smaller-scale problems, as the process of applying the heuristic to completely solve the algorithm currently requires exhaustively ranking the solutions and then exhaustively searching them; as the number of solutions grows, the process eventually takes more computational time than is saved in reaching the optimal, or a good, solution in an earlier iteration.

The coding for the proportional heuristic, for both initial solution setup and execution of exhaustive enumeration with that logic as the basis, is commented in the code included in Appendix 1. In Chapter 9, it will be shown that this heuristic not only provides a good initial solution, but also provides a better logic for small-scale exhaustive enumeration problems than regular ordination of the solution sets.

The proposed optimization formulation, which encompasses the objective function and the solution search, must be combined with the detection and prediction processes into an algorithm that can carry out the overall methodology. The construction of this algorithm is discussed in Chapter 8.

Chapter 8: The Adaptive Control Algorithm

In this chapter, the proposed adaptive control algorithm is organized, and the formulation of an executable program to carry out the algorithm is discussed.

ORGANIZATION

Thus far, three major steps in the adaptive control process have been considered: detection, prediction and optimization. These steps should be performed recursively over time as vehicles continuously arrive at the intersection, allowing the process to operate with a minimum of user-defined input. In order to do this, it will be embedded in the algorithm whose pseudo-code is outlined in Figure 8.1 below. The variables shown in Figure 8.1 have been previously defined in Chapter 7, except for the following:

- **lt**, which is lost time, the time between phases for which all approaches will be given a red state
- **ts**, the time step for each iteration of the algorithm
- **delta**, the minimum discernible difference in λ after a vehicle has arrived at the intersection

```
[Initialization]
  rt = 0
  cs = 0
  ts = (user-defined)
  lt = (user-defined)
  H = (user-defined)
  delta = (user-defined)
  ce = cs + H
  i = 1, …, m
  k = 1, …, n
  k_i = (user-defined ∀ i)
  λ_k = 0 ∀ k
  delay = ∞
[Process]
  While rt ≤ pe
```

```
            [Detection]
                [obtain input t_ij array]
                [overwrite existing t_ij array]
            [Prediction]
                [input existing t_ij array]
                [execute prediction_function]
                [output forecast t_ij array]
            [Optimization]
                [input forecast t_ij array]
                [generate λ_k set using t_ij array]
                [select λ_k using search methodology]
                [execute optimization_function)]
                [output λ_k(new), delay_new]
                If delay_new < delay
                        Then λ_k = λ_k(new)
                [return to select λ_k step until search methodology complete]
            rt = rt + ts
        [Output]
          [send λ_k to signal controller]
        [Reintialization]
          cs = cs + H*λ_1 + lt
          ce = cs + H
          delay = ∞
          λ_k = 0  ∀ k
          If k = 1
                    Then k = n
                    Else k = k – 1
          rt = cs
          [return to while statement above]
```

Figure 8.1: Pseudo-code for proposed adaptive control algorithm.

It should be noted that several steps have been added to the process: the initialization step, the output step and the reinitialization step.

The initialization step allows the phase values to be set to their starting values, with the assumption that the time point at which vehicle arrival times are input into the optimization step is at the same starting value. (This implies that there are initial detected

and predicted vehicle arrival times in place that will be updated by the algorithm.) The initialization step also allows user input of the time step, lost time and horizon variables. The lost time input does not impact the function of the algorithm, but the other two inputs do. The time step defines how often the algorithm is carried out over the course of a phase. The impact of this may be great in terms of detection and prediction; the reader is referred to Chapters 5 and 6 for further discussion of this impact. The horizon, as discussed in Chapter 7, is the time period over which the optimization is performed. As it is an element of the objective function, it can in fact be a control variable in the function and as such may be searched for an optimum. The potential for this is discussed in Chapter 9, but for this version of the algorithm, it is kept as a user-defined constant.

The output and reinitialization steps consolidate the functions carried out beforehand in the algorithm. The output step sends the optimum to the signal controller to be implemented on the phase for which traffic data was predicted. The reinitialization step updates the phase values and allows the algorithm to reiterate after it has been carried out over the course of the horizon.

This process can also be seen in the flow chart in Figure 8.2. Although the organization of the algorithm as shown in the flow chart will generally function in any situation, the specifics of the algorithm as outlined in the pseudo-code are subject to change based on a variety of factors. However, it is anticipated that the changes will be minimal, and that fine-tuning of the algorithm will allow these changes to actually be quite flexible to the point of being modular, i.e. specific blocks of the pseudo-code that carry out specific functions in the process will be interchangeable based on the required performance of that function. The factors that could warrant these types of modular alterations to the algorithm include:

- changes in the detection process (e.g. using different methods as discussed in Chapter 5)
- changes in the prediction process (e.g. using different methods as discussed in Chapter 6)
- changes in the interaction among the detection, prediction and optimization processes
- changes in the optimization process such as:
- use of other measures of effectiveness
- addition of factors to better account for driver/traffic behavior
- addition of traffic control prioritizations (e.g. enforced signal coordination/progression, transit priority, etc.)
- changes in the search procedure (e.g. different technique, parameters, initial solution, etc.)
- specific accommodations for single-intersection, arterial or network operations

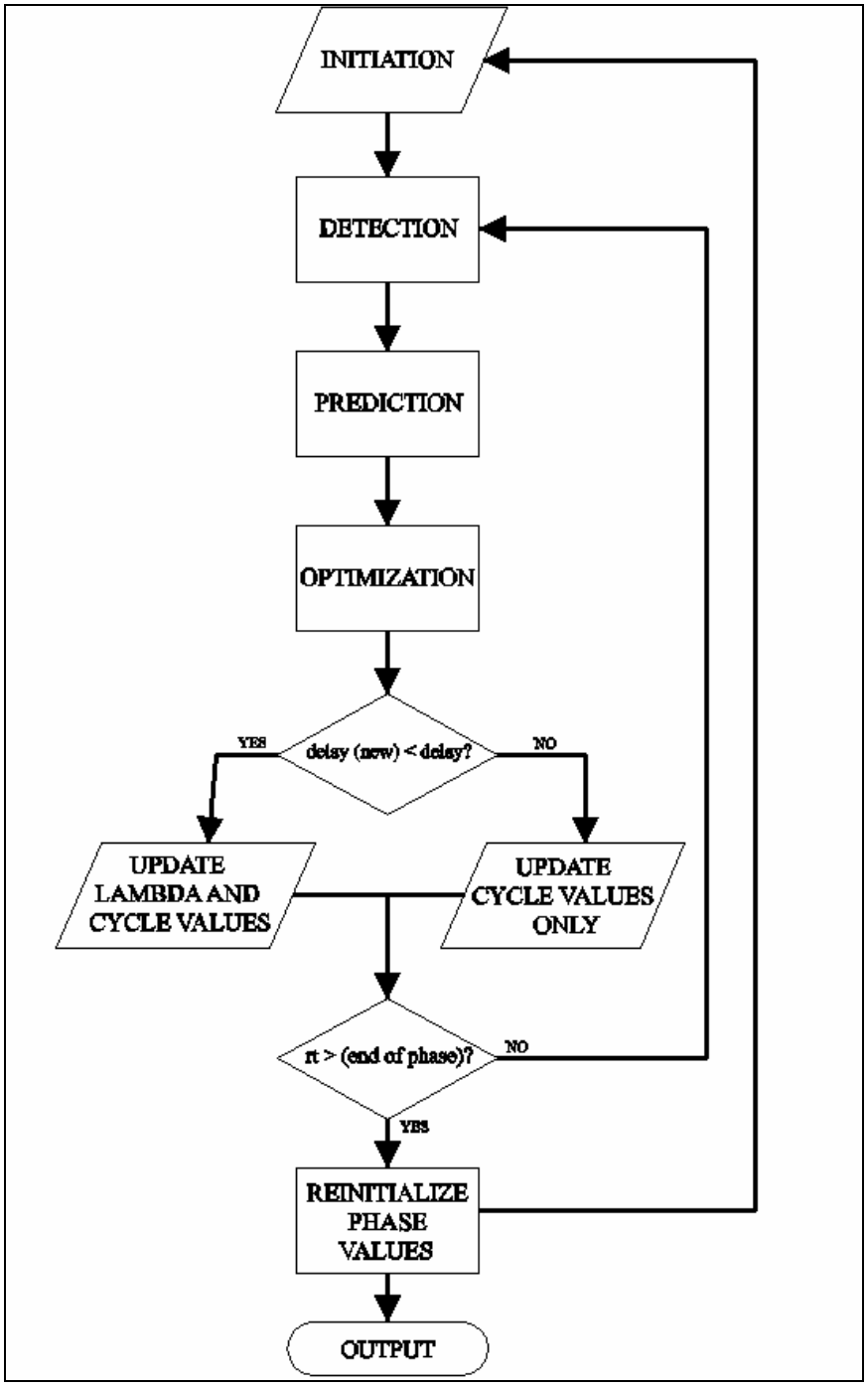

Figure 8.2. Flow chart for proposed adaptive control algorithm.

CODING OF THE ALGORITHM

In order to experiment with the algorithm and compare it to other control methodologies, the pseudo-code must first be translated into an executable program.

This was first accomplished through a Microsoft Excel spreadsheet. This platform was used to initiate implementation of the algorithm for several reasons, the main ones being the ability to clearly view the different processes undertaken by the algorithm and to allow easy troubleshooting of various aspects of the pseudo-code. This type of implementation had several drawbacks; the tediousness of data input and the use of several different workbooks for the various parts of the pseudo-code made it difficult and time-consuming to carry out longer simulation times, attempt numerous replications of the algorithm for statistical validity, and consider the algorithm's computation speed based on various problem complexities, among other desired goals. Furthermore, implementation of the metaheuristic search was not possible with the Excel implementation. However, the Excel implementation proved to be invaluable in several ways, including:

- helping to develop the final pseudo-code for the algorithm
- demonstrating the viability of the proposed adaptive control algorithm
- allowing the execution of small-scale preliminary experimentation

Detailed documentation of the spreadsheet's operation is included via the results for Experiment #1 from Chapter 9 in Appendix 3 for further review by the reader.

To address the drawbacks of the Excel implementation, the pseudo-code was next translated into the C++ programming language. The C++ implementation has the following advantages over the Excel implementation:

- allows for rapid recursive operation, both within a phase and from phase to phase
- allows execution of heuristic and metaheuristic searches
- allows automated input of arrival data from any generated file
- allows easy application of constraints
- allows tracking of iterations, optimum iteration, combinations and feasible solutions

The code for the C++ implementation is included in Appendix 1. Comments have been included to annotate "programming blocks" such as those alluded to earlier in this chapter, which may be replaced based on the functionality of the algorithm desired by the user, enhancing the algorithm's flexibility. One such functionality substitution is already in place; it is user-defined in the program execution, allowing the user to input arrival data manually rather than from an input file. Further development of the algorithm may allow much greater "user-friendliness" in making these substitutions, such as through the use of a graphical user interface.

Besides the advantages stated above, use of the C++ implementation actually allows the algorithm to interact with traffic simulation programs. Such interaction would help approximate the field implementation of the algorithm by:

- allowing acquisition of detector data from arriving vehicles to use in the prediction operation
- applying the optimum λ_k set directly to the traffic signal controller(s)
- providing comprehensive data about the MOE to be optimized.

- furnishing a test bed that allows easy comparison of the status of that MOE using the algorithm with its status for the same traffic stream using other control methodologies

As an example, CORSIM is a urban network traffic simulator that is part of the TSIS traffic simulation software package developed for use by the Federal Highway Administration that allows this interactivity. It gives a thorough output of the delay statistics of the traffic stream, allows easy use of a number of pretimed and actuated signal control schemes and is easy to setup for complex intersection and/or network configurations. However, the interface between CORSIM and the algorithm is not pre-structured in either of the two. To address this, a piece of coding called a run-time extension is required to allow CORSIM to perform this interaction. For more details on these measures, the reader is referred to Appendix 2 of this document. (Although experimentation is not carried out herein using the run-time extension, its use is being explored for further development of the algorithm.)

With a complete algorithm coded with the construction described herein, recursive operation on a set of vehicle arrival times is possible. The next step was to conduct experimentation to determine the performance of the function. The experimentation performed is discussed in Chapter 9.

Chapter 9: Experimentation and Results

In this chapter, experimentation on the algorithm will be described; the purposes of this experimentation were as follows:

- to establish the validity and proper operation of the algorithm
- to assure the ability of the metaheuristic search to find "good" (i.e. sufficiently close to the true optimum) solutions within the necessary computation time
- to demonstrate the potential of the algorithm to outperform other control methodologies, including pretimed and actuated control
- to show the ability of the algorithm to meet the above purposes under a number of situations, including variations in intersection configurations, phasing plans and demand regimes

It must be noted that the purpose of the experimentation is to demonstrate the functionality and potential effectiveness of the proposed algorithm in dealing complex traffic control situations; the experimentation was not intended to be as rigorous a test as might be carried out on an established methodology, but rather to attempt to establish the methodology.

BASICS OF THE EXPERIMENTATION

In common to all of the experiments (except where specified) are the following assumptions and treatments:

- Each approach at an intersection alternated red and green phases, and a cycle was defined as the provision of green time during one phase to each approach at the intersection, and the lost time for each of those phases.

- The lost time for each phase was fixed to 1 second.
- The delta value was fixed to 0.001 units, i.e. the each generated lambda will be (arrival time/horizon length) + 0.001.
- Traffic arrival times for the furthest upstream intersection were determined via "truncated Poisson" distribution, i.e. interarrival times were calculated using the Poisson distribution with a mean defined as the average time between arrivals for the expected demand in vehicles per hour (vph) stated for each approach in the experiment (this is just 3600 seconds per hour divided by the vph demand), with any interarrival value below 2 seconds (the generally considered minimum headway as derived by Greenshields et al. (1947)) replaced by a value of 2 seconds. Thus, this generation of interarrival times is based on the following:

$$\text{interarrival time} = \max\left[\left(\frac{H}{1-e^{x/\beta}}\right), 2\right]$$

  where x is a random number between zero and 1 and β is the average interarrival time for vehicles on the approach (equal to H divided by the demand on the approach).
- Prediction of vehicle arrivals by the algorithm based on the generated arrival times is "perfect" (i.e. vehicle arrivals input into the algorithm will be based directly on the generated interarrival times, rather than being used to predict future arrivals for use in the algorithm).

The use of the "truncated Poisson" method of generating interarrival times must be done judiciously, as observation of Table 9.1 will demonstrate. Contained in Table 9.1 are the associated statistics for set counts of generated vehicle interarrival times for

Experiment #2 over 3600 seconds, and the mean hourly demand used to generate each set.

mean hourly demand (vph)	150	300	450	600	900	1200
number of generated sets	17	23	7	22	4	1
maximum count	163	324	476	624	854	1001
minimum count	163	257	430	517	789	1001
average count	146.4	291.4	447.6	567.4	818.8	1001
standard deviation	9.48	16.41	15.50	25.52	30.50	0
coefficient of variation	6.48%	5.63%	3.46%	4.50%	3.73%	n/a

Table 9.1: Statistics on counts sets generated by "truncated Poisson" for Experiment #2.

Table 9.1 reveals several points about the "truncated Poisson" method:

- The method, on average, underestimates the hourly demand desired to be represented. The difference between the two does not increase dramatically until the demand goes above 450 vph.
- Although the coefficient of variation remains relatively steady across the demands, the standard deviation increases sharply, again for demands above 450 vph. This raises concern about the reliability of the generation method for higher demands.
- The method was unable to produce a count above the mean for demands higher than 600 vph. As the count increases, the difference between the maximum generated count and the mean used also increases.

Although the sample of generated set counts is relatively small for the higher demands, the trend towards the improper function of the method at higher demands is clear. This trend has been duly noted for the generation of interarrival times for the

Poisson distribution in general (Pignataro (1973) and May (1990)); it is exaggerated by the truncation of low interarrival times in the current method. In other words, at higher demands, "truncated Poisson" not only creates a greater number of unrealistic interarrival times, but also reduces the generated count by replacing smaller interarrival times with larger ones.

In Experiments #1 and #2, the functionality of the algorithm was tested; this did not require that the mean hourly demand be reproduced by the "truncated Poisson" method, but rather that it provide a realistic distribution of interarrival times among the vehicles in the demand, which it does. Experiment #3 is more problematic in the use of "truncated Poisson" in that the method and CORSIM will not generate the exact same number of vehicles per hour; assuring a significant difference between the CORSIM- and algorithm-generated measures of effectiveness and normalizing these measures to a per vehicle basis will help to alleviate the complication.

EXPERIMENT #1: OPERATION OF THE PROGRAMMING

The purpose of the first experiment is to ensure that the algorithm operates properly on different levels of its programming. These levels of programming are:

- Excel implementation
- C++ implementation – exhaustive enumeration
- C++ implementation – proportional heuristic
- C++ implementation – metaheuristic search

To accomplish this, a problem was devised that is simple enough to allow the true optimal solution to be found, but rigorous enough to test the proper operation for

relatively complex problems that will not be solvable on all platforms. The required characteristics of such a problem are:

- Utilization of more than two phases (this assures multiphase operation up to the current maximum of eight phases via the programming)
- Operation of at least one phase on more than one approach (this assures proper setup of the solution space)
- No matching of the phase and approach ordination, i.e. phase 1 should not operate on approach 1, etc. (this assures proper ordering of the phasing by the programming and proper operation of the recursion)
- "Simulation length" of at least one phase beyond a full cycle (this also assures proper reiteration of the algorithm)
- Multiple runs should be performed with different sets of arrival times (this will assure that there are no particular circumstances involved with a given arrival time set which allow proper algorithm operation as compared with others)

Exclusive of these requirements, the experiment can be rather compact, with relatively few vehicle arrivals in each run for which the delay should be optimized.

The configuration used for the first experiment was simple: a north-south arterial intersecting an east-west minor street. Approach 1 was the northbound approach on the arterial, Approach 2 was the southbound approach on the arterial, Approach 3 was the eastbound approach on the minor street and Approach 4 was the westbound approach on the minor street. Approaches 3 and 4 shared the first green phase of the cycle, Approach 1 had the second green phase and Approach 2 had the third and final green phase of the cycle. This configuration is diagrammed in Figure 9.1.

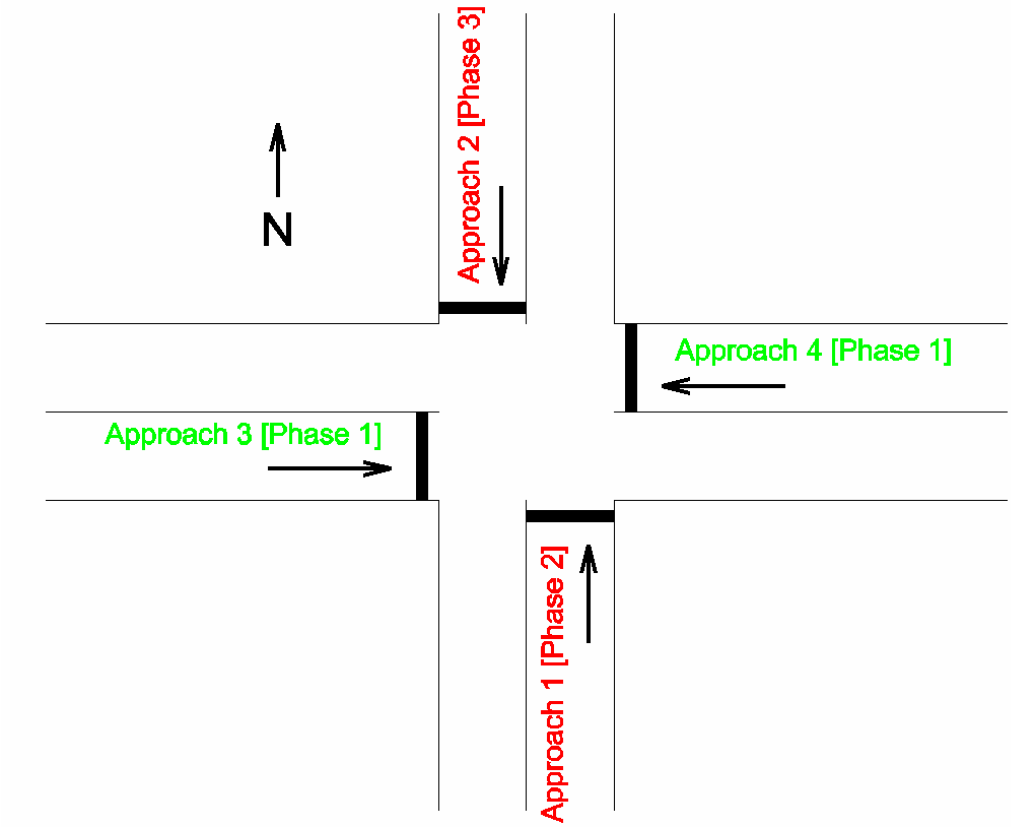

Figure 9.1: Configuration of intersection for Experiment #1.

Three runs were conducted for the first experiment with different arrival time sets. These arrival time sets, as well as a sample of the Excel implementation in resolving the first run of Experiment #1, are included in Appendix 3.

The first run used uniform arrivals for the length of the run, four iterations, with Approaches 1 and 2 both having uniform interarrival times of 20 seconds and Approaches 3 and 4 having relatively longer interarrival times of 25 and 30 seconds, respectively. As an example herein, the first Excel iteration of this run was carried out. From the overall arrival time set, the arrival times that met the constraint of being between the cycle start time, here zero seconds, and the cycle end time, here the horizon

plus the lost time accrued, or 63 seconds, were extracted for use in the optimization. These, and their associated λ_k values, are shown in Table 9.2.

	arrival times					lambdas		
approach	1	2	3	4	phase			
phase	2	3	1	1	0	1	2	3
veh.1	10	15	35	30	0	0.556	0.159	0.238
veh.2	35	45			0	0.476	0.556	0.714
veh.3	60				0	0	0.952	0
						0	0	0

Table 9.2: Arrival times and λ_k values for first run, first iteration of Experiment #1.

The next step was to generate combinations of λ_k solutions; in this case, there were 4^3, or 64, possible ones, although a number of these were clearly duplicated sets. Many of them also did not meet another constraint of the optimization, which is that each lambda should be greater than or equal to its predecessor. In fact, after these two conditions, only the following 8 of the 64 combinations were unduplicated feasible solutions (in order of their appearance in ordinal exhaustive enumeration):

- combination 6: [0.557 0.557 0.715]
- combination 22: [0.477 0.557 0.715]
- combination 33: [0.000 0.160 0.239]
- combination 34: [0.000 0.160 0.715]
- combination 38: [0.000 0.557 0.715]
- combination 45: [0.000 0.000 0.239]
- combination 46: [0.000 0.000 0.715]
- combination 47: [0.000 0.000 0.000]

From these, the delay for each vehicle was calculated, based on the appropriate phasing, dummy variables, etc. As an example, the results for combination 22 are shown in Table 9.3.

Delay ID	x_{ij}	k_i	before green delay	$(1 - x_{ij})$	after green delay	y_{ij}	$(1 - y_{ij})$	Stopped delay	Queue delay
$delay_{11}$	1	2	30.06 – 10	0	63 – 10	0	1	20.06	2
$delay_{12}$	0	2	30.06 – 35	1	63 – 35	1	0	0	0
$delay_{13}$	0	2	30.06 – 60	1	63 – 60	0	1	3	2
$delay_{21}$	1	3	35.06 – 15	0	63 – 15	0	1	20.06	0
$delay_{22}$	0	3	35.06 – 45	1	63 – 45	1	0	0	0
$delay_{31}$	0	1	0 – 35	1	63 – 35	0	0	28	0
$delay_{41}$	0	1	0 – 30	0	63 – 30	1	0	0	0

Table 9.3: Delay table for first run, first iteration of Experiment #1.

Thus, the sum of stopped delays for this combination was 75.12 vehicle-seconds. The combinations each yielded the following results for their total stopped delays:

- combination 6: 55.18 vehicle-seconds
- combination 22: 75.12 vehicle-seconds
- combination 33: 110 vehicle-seconds
- combination 34: 92 vehicle seconds
- combination 38: 86.06 vehicle-seconds
- combination 45: 163 vehicle-seconds
- combination 46: 145 vehicle-seconds
- combination 48: 211 vehicle-seconds

There were no carryover delays, and so combination 6 was considered the optimum solution. Because λ_1 was 0.557 for this combination, phase 1 ended at cs + λ_1H + lt, which is 0 + (0.557*63), or 36.06 seconds. The actual experienced stopped delay

after this phase was 25.12 seconds, because this delay was accrued before the start of phase 2 for this combination. There was also a delay carried over for the vehicle arriving at Approach 2 at 15 seconds, because it did not receive a green phase during the next iteration, which started at cs = (35.06 + lt) = 36.06 seconds, causing this vehicle to not be considered in the next optimization. It was 20.06 seconds after the first iteration, but subject to change depending on the outcome of the second iteration.

The outputs for the C++ implementations included in Appendix 3 show that these calculations have been reproduced in the first iteration of each of them. Figure 9.2 shows a screen capture of the output of the C++ exhaustive enumeration implementation with the results replicated from those yielded by the manual execution.

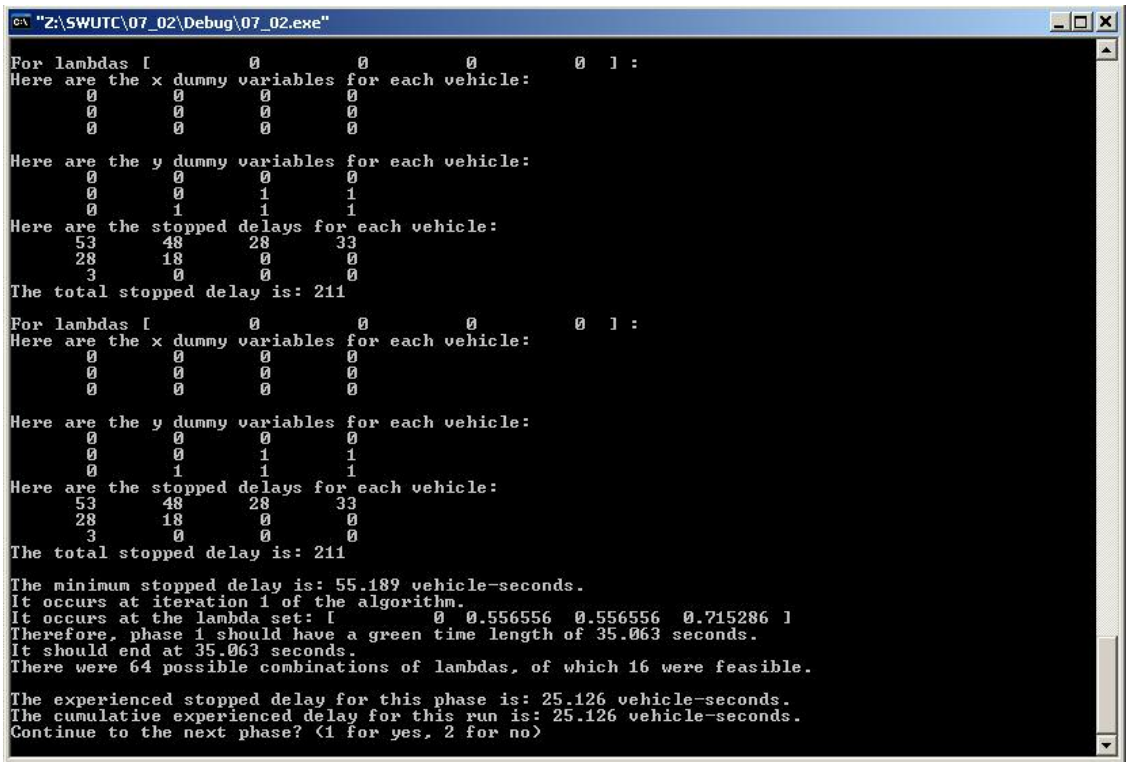

Figure 9.2: Screen capture of exhaustive enumeration results of first run, first iteration of Experiment #1.

The next iteration captured a vehicle arrival time set between 36.06 and (36.06 + 63), or 99.06 seconds. After this iteration, the cycle was shifted again to start at the end of the "new phase 1", and this continued until 4 iterations were carried out for each implementation to be tested. The resulting cumulative values are provided in Table 9.4.

Iter.	attribute	Excel	exhaustive	proportional	metaheuristic
1	lambda set	0.557, 0.557, 0.715	0.557, 0.557, 0.715	0.557, 0.557, 0.715	0.557, 0.557, 0.715
	stopped delay	55.273	55.189	55.189	55.189
	exp. delay	25.182	25.126	25.126	25.126
	phase length	35.091	35.063	35.063	35.063
2	lambda set	0.000, 0.143, 0.540	0.000, 0.143, 0.540	0.000, 0.143, 0.540	0.000, 0.143, 0.540
	stopped delay	100.240	100.252	100.252	100.252
	exp. delay	0.000	0.000	0.000	0.000
	phase length	0.000	0.000	0.000	0.000
3	lambda set	0.127, 0.524, 0.762	0.127, 0.524, 0.762	0.127, 0.524, 0.762	0.127, 0.524, 0.762
	stopped delay	37.132	37.126	37.126	37.126
	exp. delay	0.000	0.000	0.000	0.000
	phase length	8.001	8.000	8.000	8.000
4	lambda set	0.381, 0.619, 0.937	0.381, 0.619, 0.917	0.381, 0.619, 0.917	0.381, 0.619, 0.917
	stopped delay	28.180	28.189	28.189	28.189
	exp. delay	10.063	10.063	10.063	10.063
	phase length	24.003	24.000	24.000	24.000

Table 9.4: Results of the first run of Experiment #1 for all platforms.

The results in Table 9.4 are equivalent (excluding rounding errors in the Excel implementation), thus all platforms are shown to be operative for the first run of Experiment #1. The overall length of the first three phases, and thus the first complete cycle, was 45.12 seconds and the final experienced stopped delay for the run was 35.19 vehicle-seconds. The equivalence of these values across the platforms reinforces their

proper operation; the importance of the values themselves is in comparison with other methodologies, a task which was carried out in Experiment #3.

As stated earlier, it was desirable to attempt runs with other arrival time sets to ensure that there were no peculiarities about the time set used in the first run allowing the C++ implementations to replicate the operation of the Excel implementation. Two more runs were thus attempted; the first of these (hereafter run #2) was made in a similar manner to run #1, but essentially with higher demand placed on the minor street than on the arterial. The second (hereafter run #3) was carried out using arrival times generated by the "truncated Poisson" method using the following hourly demands:

- Approach 1: 200 vehicles per hour
- Approach 2: 100 vehicles per hour
- Approaches 3 and 4: 50 vehicles per hour

The demands were kept relatively low to allow Excel implementation for confirmation. The results of runs 2 and 3 are shown in Table 9.5.

Run	attribute	Excel	exhaustive	proportional	metaheuristic
2	phase 1 λ	0.318	0.318	0.318	0.318
	phase 2 λ	0.143	0.143	0.143	0.143
	phase 3 λ	0.301	0.302	0.302	0.302
	phase 4 λ	0.381	0.381	0.381	0.381
	iter. 4 stopped delay	71.196	71.315	71.315	71.315
	cum. exp. delay	45.202	45.315	45.315	45.315
	cycle length	51.006	51.069	51.069	51.069
3	phase 1 λ	0.000	0.000	0.000	0.000
	phase 2 λ	0.478	0.478	0.478	0.478
	phase 3 λ	0.212	0.211	0.211	0.211
	phase 4 λ	0.000	0.000	0.000	0.000
	iter. 4 stopped delay	242.330	242.141	242.141	242.141
	cum.exp. delay	18.294	18.286	18.286	18.286
	cycle length	46.470	46.443	46.443	46.443

Table 9.5: Results for key attributes of runs 2 and 3 of Experiment #1.

The results from both Table 9.4 and 9.4 demonstrate the proper operation of all the implementations of the algorithm for a variety of arrival time sets. The variations in demand for these arrival sets demonstrates, in a preliminary sense, the ability of the algorithm to operate in different phasing conditions as well; this is because the operation of the algorithm consists essentially of:

- Translating a set of inputs into a set of independent variables and an associated ordering
- Finding a solution set that minimizes a function given that set and ordering, regardless of the values of those variables

EXPERIMENT #2: OPERATION OF THE HEURISTIC AND METAHEURISTIC SEARCHES

The purpose of the second experiment is twofold:

- It should be established that the heuristic and metaheuristic searches can consistently outperform exhaustive enumeration of the solution space, in terms of computation time and/or the number of iterations, in finding the true optimum solution
- It should also be established that the heuristic and (particularly) metaheuristic searches can consistently find a solution that is sufficiently close to the true optimum solution within some operational limit (i.e, a given number of iterations or a given computation time)

The accomplishment of these purposes involves devising a problem for which the true optimum solution can be found with certainty, by the C++ implementation with exhaustive enumeration. The problem would then be applied to the C++ implementations with the proportional heuristic and the metaheuristic search for a number of generated arrival time sets.

For the first purpose, as the program stores and then outputs the iteration at which the best current solution was found, it is be compared with the value for the exhaustive enumeration statistically over a number of runs to assure that there is a significant performance improvement. For the second purpose, as the program stores and then outputs the best current solution, it is again compared over a number of the same runs with the operational limit imposed to ensure that it is consistently close enough statistically to the true optimal solution to merit its use.

For the experiment, a total of 8 arrival time sets were input into the algorithm for each of the platforms, with a variety of demands and phasing and intersection configurations to attempt to cover a wide range of scenarios that might be encountered in the field. The basic geometry of the intersection conformed to that used in Experiment

#1, but with the addition for some runs of left turning lanes on each leg with independent sets of arrival times from the major flow. Approach 1 added a turning lane labeled Approach 5, Approach 2 added Approach 6, Approach 3 added Approach 7 and Approach 4 add Approach 8, so that the final configuration with all eight approaches was as shown in Figure 9.3.

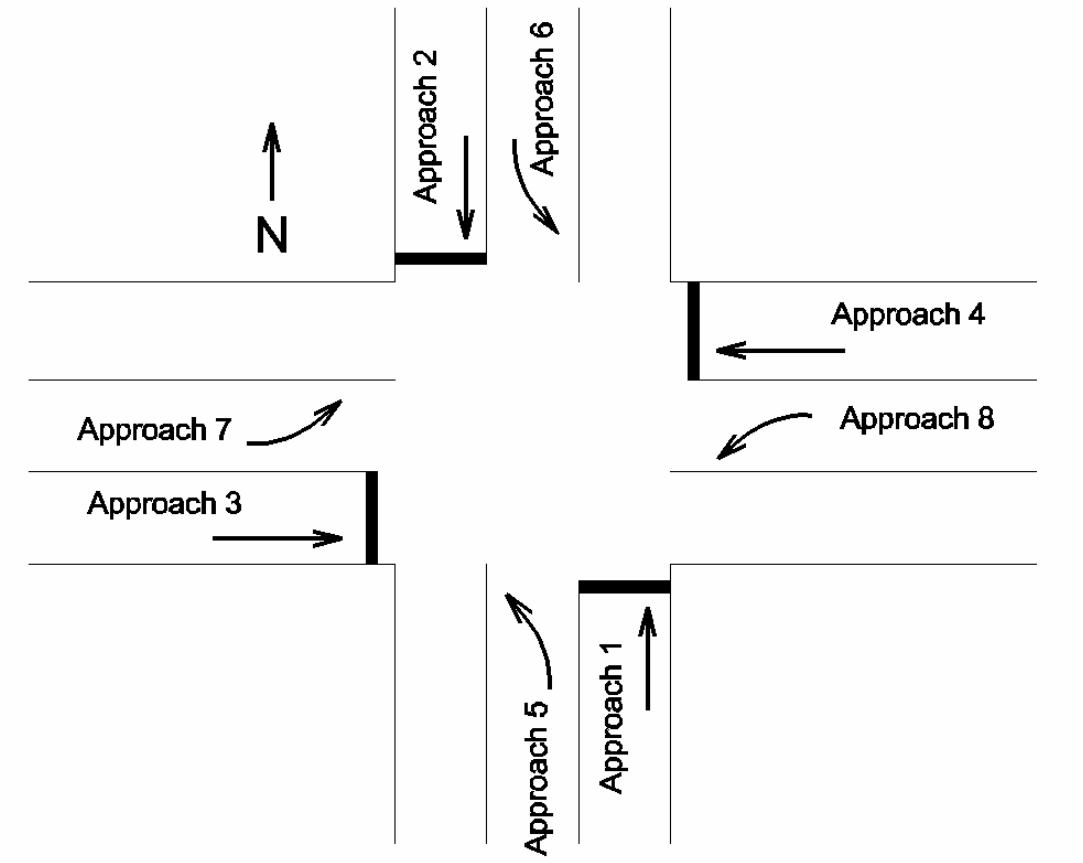

Figure 9.3: Configuration of intersection for Experiment #2.

The intersection configuration, phasing plan, and hourly demand for each leg used for each run of Experiment #2 are shown in Table 9.6. As indicated in Chapter 7, use of the proportional heuristic is not preferable for larger-scale problems; runs 5 through 8 are in the realm of this type of problem, with the number of combinations exceeding 10^5 (and

usually no less than 10^6) for nearly every recursion of the algorithm. Thus, its use will be limited to runs 1 through 4, while runs 5 through 12 will compare only the exhaustive enumeration with the metaheuristic search.

		Approach							
attribute	run	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
phase	1 to 4	2	3	1	1	n/a	n/a	n/a	n/a
demand (vph)	1	900	600	300	300	n/a	n/a	n/a	n/a
	2	600	600	600	600	n/a	n/a	n/a	n/a
	3	300	300	600	150	n/a	n/a	n/a	n/a
	4	1200	300	300	300	n/a	n/a	n/a	n/a
phase	5 to 8	2	1	3	4	5	5	n/a	n/a
demand (vph)	5	900	600	300	300	150	150	n/a	n/a
	6	600	600	600	600	300	300	n/a	n/a
	7	300	300	600	300	300	300	n/a	n/a
	8	900	450	450	450	150	150	n/a	n/a

Table 9.6: Approaches, demands and phasing for Experiment #2.

The runs were allowed to continue until the phase end reached 3600 seconds (1 hour), and arrival times were generated to no less than 3670 seconds for each leg to allow an ample time window for the horizon beyond the stopping point. To ensure further variation among the runs, the arrival times were generated anew for each run. For each run, the following statistics were recorded for each implementation: number of recursions, final cumulative experienced stopped delay and average iteration to solution. Also recorded was the first recursion at which optimality was not achieved for a non-exhaustive search, if this occurred, and the λ_1 value, and total and experienced stopped delay for each at that recursion. These results are recorded in Table 9.7.

pltfm.	attribute	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8
Exh.	avg. iter. #	178	247	35	116	44	1011	154	40
	# recursions	160	162	262	191	265	190	427	271
	std. deviation	126.24	159.87	34.75	93.07	75.38	1493.22	257.45	51.37
	cum. exp. del	6766.4	8222.8	3704.8	5841.2	2937.64	6543.95	3631.82	2970.30
	λ_1 at deviation	0.319	0.314	0.577	0.216	0.248	0.501	0.025	.061
	exp. delay at dev.	42.67	23.85	55.71	24.07	14.53	33.03	0.00	0.79
	stop delay at dev.	438.47	386.20	390.13	262.34	322.20	803.975	308.95	350.13
Prop.	avg. iter. #	74	78	26	63	n/a	n/a	n/a	n/a
	# recursions	167	179	260	200	n/a	n/a	n/a	n/a
	std. deviation	23.06	19.87	26.27	31.64	n/a	n/a	n/a	n/a
	cum. exp. del	6416.1	7376.9	3757.5	5653.7	n/a	n/a	n/a	n/a
	recursion at dev.	2	1	2	3	n/a	n/a	n/a	n/a
	λ_1 at deviation	0.319	0.314	0.577	0.216	n/a	n/a	n/a	n/a
	exp. delay at dev.	42.67	23.85	55.71	24.07	n/a	n/a	n/a	n/a
	stop delay at dev.	445.81	480.04	147.06	282.18	n/a	n/a	n/a	n/a
Meta.	avg. iter. #	102	146	29	81	128	240	104	106
	# recursions	161	159	267	191	270	282	451	274
	std. deviation	65.05	95.38	42.09	73.86	166.46	137.34	122.31	152.89
	cum. exp. del	6773	7815.64	3463.42	6041.55	2739.42	4798.21	3334.32	3093.80
	recursion at dev.	2	1	1	6	1	1	1	2
	λ_1 at deviation	0.319	0.759	0.485	0.015	0.248	0.289	0.025	.061
	exp. delay at dev.	42.67	136.01	24.46	0.00	14.54	18.80	0.00	0.79
	stop delay at dev.	445.81	396.40	246.96	193.94	324.19	903.08	313.93	351.44

Table 9.7: Results of Experiment #2.

Several points should be noticed about the results in Table 9.7, indicating that, in fact, the non-exhaustive searches can outperform the exhaustive search and can yield suitable solutions given an operational constraint:

- On the average, the non-exhaustive searches tend to obtain a solution at earlier iterations than the exhaustive search. This is partially due to restriction of the number of iterations to a maximum of 100 for the proportional heuristic and a maximum of 500 for the metaheuristic search. Despite this, the non-exhaustive searches needed to go to the maximum iteration for less than 10 percent of the recursions. This seems to be an indication of the relatively better performance of the non-exhaustive searches. As implied earlier, this evidence of improvement in performance is diminished for the proportional heuristic by the longer

computation time required. Thus, it should not be considered to be advantageous to use it for any greater complexities than the ones tested herein.

- The standard deviations of the iterations for the non-exhaustive searches are generally smaller than those of the exhaustive enumeration. This is an indication, not only of better performance, but of greater reliability of the non-exhaustive searches in reaching solutions at earlier iterations than the exhaustive searches.
- The differences between the search attributes of the exhaustive and non-exhaustive searches at the recursion that the non-exhaustive searches deviate is typically less than 10 percent, and the values are actually often equal for the crucial values of experienced stopped delay (the measure of effectiveness) and λ_1 (the value actually used to set the signal control). This is indicative of the soundness of the solutions found by the non-exhaustive searches.
- The difference between the values of the cumulative experienced stopped delay for the exhaustive and non-exhaustive searches is also typically less than 10 percent (often less than 5 percent). This reflects the soundness of the solutions found by the non-exhaustive searches for multiple deviating recursions over the length of the run.

There are some instances where the metaheuristic search results in an attribute that is significantly deviant from the result of the exhaustive enumeration (e.g the number of recursions for Run #6). This is a result of the relatively random nature of the jumps between solutions in the proportional solution space. The key to evaluating the performance of the metaheuristic search is its ability to produce measures of effectiveness (cumulative experienced stopped delay) relatively close to (and even improved over, in some cases) the exhaustive enumeration for complex problems with a lower number of

iterations and/or computation speed Further development may result in an even better performing search, but it is safe to say from these results that the metaheuristic search is more effective, especially considering the reduced computation time that the metaheuristic search consumes in relation to either the exhaustive enumeration or the proportional heuristic.

EXPERIMENT #3: COMPARISON WITH OTHER CONTROL METHODOLOGIES

The purpose of the third experiment is to demonstrate the viability of the use of the proposed adaptive traffic signal control algorithm as an alternative to the traditional traffic signal control methodologies of pretimed and actuated control. In order to accomplish this, simulation will be carried out using CORSIM, the traffic simulation program discussed in the coding section of Chapter 8.

The intersection configurations, phasing plans and arrival time data were the same as those used in Experiment #2. These were be input into CORSIM to allow the simulation to represent as closely as possible the runs carried out in Experiment #2. The key difference was in the arrival time sets; the hourly demands from Experiment #2 were input into CORSIM, which uses its own exponential distribution to assign arrival times for vehicles from this demand. While this distribution was quite similar to that of the arrival time sets generated for Experiment #2, the differences do not allow for a completely direct comparison, but rather a parallel between the CORSIM and proposed algorithm runs. Proof of improvement lies in finding a significant difference in key measures of effectiveness over the course of the runs for the two methodologies.

The phasing for CORSIM will be based on Webster's method for pretimed control as described in Pignataro (1973). The phasing for the actuated control will be based on semi-actuation as described in Pignataro (1973). (Semi-actuation implies that only the east-west minor street will have a detector for actuated control for Runs 1 through 4, and only the turning lanes will have one for Runs 8 through 12; the other approaches will operate as pretimed signals with green times distributed via Webster's method.) The calculations for these methods are outlined in Appendix 4.

The results of the experiment comparing the measures of effectiveness are shown in Table 9.8. The reductions in delay are quite significant for both actuated and pretimed control as compared with the proposed algorithm, but there are several points to bear in mind in considering these results:

- The algorithm was carried out with ideal conditions, including predetermined arrival times, very little compensation for driver behavior patterns, and a myriad of other considerations that have yet to be made.
- The traffic configuration and arrival patterns were not ideal for the CORSIM applications, particularly the semi-actuated control, which suffered badly in the 5-phase control of runs 5 through 8.
- The majority of delays for the CORSIM applications were accrued in approaches performing turning movements. Although traditional control measures were used to attempt to account for the turning traffic, controls for these movements are often further optimized to reduce delay.

Despite these points, and the fact that the proposed control will be highly unlikely to perform nearly as well in the field, parallels in comparison (phasing, mean traffic levels, intersection configuration) from which it may be concluded that the algorithm is

able to outperform the traditional control used were held. The results herein speak to the potential of the proposed methodology to greatly reduce delay by accounting for changes in demand and phasing configuration, and by addressing the arrivals of individual vehicles as they use the intersection rather than dealing with volumes of traffic over intervals without addressing changes in demand.

		Run 1	Run 2	Run 3	Run 4	Run 5
algorithm	stopped del. (veh-sec.)	6766	8222	3705	5841	2938
	total vehicles	1898	2262	1292	1895	2326
	avg. delay/veh. (sec.)	3.56	3.63	2.87	3.08	1.26
CORSIM pretimed	stopped del. (veh-sec.)	55950	38748	19356	42858	221478
	total vehicles	2032	2392	1346	1125	1727
	avg. delay/veh. (sec.)	27.53	16.20	14.51	38.10	128.24
	% reduction in avg.	87.07%	77.59%	80.22%	91.92%	99.02%
CORSIM actuated	stopped del. (veh-sec.)	41586	36048	31728	40362	256986
	total vehicles	2096	1203	1041	1798	663
	avg. delay/veh. (sec.)	19.84	29.97	30.48	22.45	387.60
	% reduction in avg.	82.06%	87.89%	90.58%	86.28%	99.67%
		Run 6	Run 7	Run 8	over all runs	
algorithm	stopped del. (veh-sec.)	6544	3632	2970	40618	
	total vehicles	2882	2228	2482	17265	
	avg. delay/veh. (sec.)	2.27	1.63	1.20	2.30	
CORSIM Pretimed	stopped del. (veh-sec.)	228522	188446	231450	1026808	
	total vehicles	1772	1925	1460	13779	
	avg. delay/veh. (sec.)	128.96	97.89	158.53	74.52	
	% reduction in avg.	98.24%	98.33%	99.24%	96.91%	
CORSIM actuated	stopped del. (veh-sec.)	252558	245910	254076	1159254	
	total vehicles	914	390	386	8491	
	avg. delay/veh. (sec.)	142.53	630.54	658.23	136.53	
	% reduction in avg.	98.41%	99.74%	99.82%	98.32%	

Table 9.8: Results of Experiment #3.

Overall, although somewhat rudimentary, the experimentation bears out the proper operation and performance of the function. Further research into and development

of the algorithm can lead to broader experimentation that would provide even more insight, particularly into the comparative performance of the methodology to other techniques. Consideration of that research, development and experimentation is made in Chapter 10.

Chapter 10: Summary, Conclusions and Further Research

In developing an adaptive traffic signal control methodology that advances the concept of adaptive in control in various ways, previous implementations were drawn upon, and new ideas were applied. The purposes of applying these new ideas include:

- Accounting for advancements in all steps in the adaptive control process
- Addressing drawbacks in previous methodologies
- Dealing with driver behavior in utilizing traffic signal control
- Allowing flexibility for performance in a variety of scenarios
- Allowing user control of a number of parameters
- Providing a transparent structure that can permit addition, removal or exchange of processes
- Creating a competitive methodology relative to traditional forms to traffic signal control
- Increasing speed of computation to allow use in complex applications

The proposed traffic signal control methodology meets these purposes, as demonstrated through their consideration in the development and its description and the performance of the methodology in experimentation. Accomplishing this in a proposed methodology rather than enhancing an existing methodology was necessary, and also significant, but the algorithm should not be considered as a finished product, but rather as a construct in its infancy. There are a number of specific measures which can be taken to further address each of the purposes listed above.

To address the issues brought about by other steps in the adaptive control process, they should be incorporated into the algorithm in the following ways, among others:

- through the use of some of the prediction measures mentioned in Chapter 6 in conjunction with actual field data to determine their accuracy and the effect on the algorithm
- by simulating different detector configurations to see their effect on the prediction process and the algorithm though their ability to accurately measure traffic data

To continue dealing with drawbacks in previous methodologies, the algorithm must be able to function in similar environments. This requires eventual implementation in a field setting. Eventual field trials in settings as described in Chapter 4 will help determine which areas of concerns from those implementations have been addressed herein and which still need to be addressed.

To further deal with driver behavior in the algorithm, the following issues must be considered in enhancing the formulation:

- The effect of short phase lengths on queue startup and queue clearance
- The effect of conflicting movements for approaches sharing a phase
- The effect of multilane configurations and resulting passing movements

To allow further flexibility for various scenarios, the following should be considered:

- Testing of the algorithm on arterial and then network configurations
- Implementing the ability to have one approach utilize two phases (e.g. an approach using a straight-only phase and a straight-and-left-turn phase)

To provide greater user control over parameters and "functional blocks" of the algorithm, a better user interface should be developed. While it is currently possible for a user to find sections of the annotated code to remove and replace to suit their preference, a graphical user interface would encourage the user to adapt the methodology to their needs and, in fact, improve its performance.

To help create an even more competitive methodology, direct comparisons with other methodologies must be made through simulation. The run-time extension, applied to the methodology as described in Appendix 2, will allow this kind of comparison through CORSIM, permitting enhancement of the methodology to specifically address the outcomes of the comparison.

To enhance the speed of the computations to allow the use of the algorithm in more complex implementations, the following steps may be taken:

- Addition of constraints or other elements, such as adaptive tabu tenure or clustering recognition, to the metaheuristic search to allow more efficient solution selection
- Allowing the switching on and off of phase-by-phase operation to allow cycle-based optimization on lower demand, lower variability, or time-constrained applications
- Releasing the phasing order constraints, particularly in the metaheuristic search, to allow exploration of the non-feasible space for faster or more improved results
- Incorporating phase skipping when no vehicle arrivals are anticipated

Despite the breadth and depth of possibilities for enhancing the proposed adaptive control methodology, it is in its present state, as described herein, a viable, stand-alone

traffic signal control. It optimizes traffic signal phasing for stopped delay based on the actual arrival times of vehicles rather than their presence or volume in any interval, and it performs this function on a phase-by-phase basis, rather than maintaining the optimized horizon over an entire cycle. It incorporates both a heuristic and a metaheuristic search that are capable of finding suitable and/or optimal solutions for phase timing more quickly than ordinal exhaustive enumeration. It outperforms traditional methods of traffic signal control in a parallel comparison. Finally, it addresses the issues raised in introducing the idea of advancing the concept of adaptive traffic control.

Appendix 1: C++ Coding of the Algorithm

Code for Exhaustive Enumeration and Proportional Heuristic

```
/*#include <stdlib.h>
//#include "stdafx.h"
#include <iostream>
using std::ios;
#include <fstream>
using std::ifstream;
#include <iomanip>
#include <string>
#include < math.h>
#include <new>
using namespace std;

using std::string;
#using <mscorlib.dll>
*/
//Opening of header files to be inserted into program
#include <iostream.h>
#include < iomanip.h>
#include <fstream.h >
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <new.h>

int main()
{
//inventory of introduced variables
   const int mrow = 60;
   const int mcol = 8;
   int arr, i, j, m, no, count, count2, count3, count6, position, iterate, iter_opt, manin, lt, phase, count_iter;
   double at[mrow][mcol], arr_time[mrow][mcol], lam[mrow][mcol], stop_del[mrow][mcol],
lambda[mrow][mcol], lam_sol[mcol], lam_out[mcol], lam_combo[mrow], sum_at[mcol],
sum_at_temp[mcol];
   int xdummy[mrow][mcol], ydummy[mrow][mcol], ki[mrow][mcol], k[mcol],  cntr[mcol],
cntr_temp[mcol];
   double h, l_inc, delay, total_del, total_stop_del, exp_del, cum_del, cs, rt, ts, pe, n, lam_max,
diff_sum_min, diff_sum_temp, diff_sum, NC, prop_rank, co_del;

   ifstream Input;


 /*   ofstream Output;
   Output.open("outfile.txt");*/

/*    for (i=0;i<mrow;i++){
       for (j=0;j<mcol;j++){
          cout<<stop_del[i][j]<<"\t";
       }
```

```
      cout<<"\n";
   }
*/

//initialization
   cs = 0;
   rt = 0;
            phase = 1;
   no = 1;
   ts = 0;
            cum_del = 0;
   for (i=0; i < mcol; i++){
      k[i] = 0;
                        cntr[i] = 0;
                        sum_at[i] = 0;
                        cntr_temp[i] = 0;
                        sum_at_temp[i] = 0;
   }

//user definition of variables
   cout << "Welcome to the PAD program for adaptive traffic signal control! \n";
   cout << "I will find the optimum lambda and horizon length for your signal!\n";
   cout << "What would you like the horizon length (assumed cycle length) to be?\n";
   cin >> h;
   cout << "What would you like the minimum discernible lambda increment to be?\n";
   cin >> l_inc;
   cout << "What would you like the lost (all-red) time to be for each phase?\n";
   cin >> lt;
/*    cout << "What time step would you like the program to reiterate over (typically 0.5 to 5 seconds)?\n";
   cin >> ts;*/
   cout << "How many approaches do you have to the intersection?\n";
   cin >> m;
   cout << "How many phases would you like to use for the cycle?\n";
   cin >> n;
   cout << endl;
            cout << "The assumed cycle length will be adjusted for lost time to " << h + (n * lt) << "
seconds.\n";
   cout << "Would you like to manually input arrival times [type '1'] or get them from the 'arr_time.txt' file
[type '2']?\n";
   cin  >> manin;
   cout << endl;
   for (count6=0; count6<m; count6++){
   cout << "What phase is approach " << count6 + 1 << " being assigned to? \n";
   cin >> k[count6];
   }


//phase-to-phase process condition (user-defined)
while (no == 1){

   for (j=0; j < mrow; j++){
      for (i=0; i < mcol; i++){
         arr_time[j][i] = 0;
```

```
                                lam[j][i] = 0;
                                lambda[j][i] = 0;
                                ki[j][i] = 0;
        }
    }

    for (i=0; i < mcol; i++){
        lam_sol[i] = 0;
    }

    pe = cs + h;

//in-phase process condition (automatic)
//while (rt < pe){

    arr = 0;
    count6 = 0;
    iterate = 0;
            rt = rt + ts;
            cout << endl;
    cout << "The cycle start time is now " << cs << " and the nominal cycle end time is " << (cs+h) << ".\n";

//arrival time setup for manual input
    if (manin == 1){
    cout << "Please enter arrival times within the range of the cycle start and end times.\n";
    while (count6 < m){
                        cout << "For approach " << count6 + 1 << ": \n";
        cout << "How many vehicles are arriving? \n";
        cin >> j;
        cout << "Enter the arrival time of each vehicle: \n";
            for (count = 0; count < j; count++){
                cout << "Vehicle " << (count + 1) << ": ";
                cin >> arr_time[count][count6];
                ki[count][count6] = k[count6];
            }
        cout << endl;
        count6++;
        if (j > arr)
            arr = j;
    }
    cout << endl;
    }

//arrival time setup for file input
    if (manin == 2){

        Input.open("arr_time.txt");

        for (j=0;j<mrow;j++){
            for (i=0;i<mcol;i++){
                Input>>at[j][i];
            }
        }
```

```
      Input.close();
/*      for (j=0;j<mrow;j++){
        for (i=0;i<mcol;i++){
           cout<<setw(8)<<at[j][i];
        }
        cout << endl;
      }*/
      count = 0;
      count2 = 0;
      for (i=0;i<mcol;i++){
         for (j=0;j<mrow;j++){
            if (at[j][i] > (cs)){
               if (at[j][i] <= (cs + (n*lt) + h)){
                  arr_time[count][count2] = at[j][i];
                  ki[count][count2] = k[count2];
                  count++;
               }
            }
         }
      count=0;
      count2++;
      }

      for (i=0;i<mcol;i++){
         count6 = 0;
         for (j=0;j<mrow;j++){
            if (arr_time[j][i] > 0){
               count6++;
            }
         if (count6 > arr)
            arr = count6;
         }
      }
   }

//lambda setup based on vehicle arrival times
   for (j = 0; j < arr; j++){
      for (i = 0; i < m; i++){
      if (arr_time[j][i] > 0)
         lam[j][i] = ((arr_time[j][i] - cs) / (h + (n*lt))) + l_inc;
      }
   }

//arrangement of lambda solution space based on phasing
   count2 = 0;
   lam_max = 0;
   for (count = 1; count <= n; count++){
      for (j = 0; j < arr; j++){
         for (i = 0; i < m; i++){
            if (lam[j][i] > 0)
               if (ki[j][i] == count)
                  lambda[count2][count] = lam[j][i];
               if (ki[j][i] == count)
```

```
                count2++;
            if (count2 > lam_max)
                                                    lam_max = count2;
        }
      }
      count2 = 0;
   }

//display of independent variables for confirmation
   cout << "Here is the arrival time matrix for this cycle:" << endl;
   for (j = 0; j < arr; j++){
      for (i = 0; i < m; i++){
         cout << setw(8) << arr_time[j][i];
      }
      cout << endl;
   }
   cout << endl;

   cout << "Here are the phases for each vehicle:" << endl;
   for (j = 0; j < arr; j++){
      for (i = 0; i < m; i++){
         cout << setw(8) << ki[j][i];
      }
      cout << endl;
   }
   cout << endl;

   cout << "Here is the lambda matrix to be used:" << endl;
   for (j = 0; j <= lam_max; j++){
      for (i = 0; i <= n; i++){
         cout << setw(10) << lambda[j][i];
      }
      cout << endl;
   }
   cout << endl;

   delay = 1000000;

//solution selection for exhaustive enumeration
/*    int *index;
   index=new int [n+1];

   for (i = 0; i <= n; i++){
      index[i] = 0;
   }

   double NC = pow((lam_max+1), n);

   for ( count = 1; count <= NC; count++ ){

   if ( count == 1 ){
      for ( count2 = 1; count2 <= n; count2++ )
         index[count] = 0;
```

```
      }
   else {
      position = n;
      while ( index[position] == lam_max )
         position--;
      index[position]++;
      for ( count3 = position+1; count3 <= n; count3++ )
         index[count3] = 0;
           }

           for ( count2 = 0; count2 <= n ; count2++){
      lam_sol[count2] = lambda[index[count2]][count2];
   }
*/
```

//solution selection for proportional heuristic

```
   int *index;
   index=new int [n+1];

   for (i = 0; i <= n; i++){
      index[i] = 0;
   }

   NC = pow((lam_max+1), n);

   diff_sum=20;
           diff_sum_min=0;

           double total_veh=0;
           double veh_count[mcol], prop_lam[mcol];
           for (i=0; i<=mcol; i++){
                      veh_count[i] = 0;
                      prop_lam[i]=0;
           }

           for (j=0; j<=lam_max; j++){
                      for (i=0; i<=n; i++){
                                 if (lambda[j][i] > 0){
                                            veh_count[i]++;
                                            total_veh++;
                                 }
                      }
           }

   for (i = 0; i <= mcol; i++){
      prop_lam[i] = (veh_count[i])/total_veh;
   }

for (count_iter = 1; count_iter <= NC; count_iter++ ){

   for ( count = 1; count <= NC; count++ ){

   if ( count == 1 ){
```

```
      for ( count2 = 1; count2 <= n; count2++ )
         index[count] = 0;
      }

   else {
      position = n;
      while ( index[position] == lam_max )
         position--;
      index[position]++;
      for ( count3 = position+1; count3 <= n; count3++ )
         index[count3] = 0;
           }

           diff_sum_temp = 0;

           for ( count2 = 0; count2 <= n ; count2++){
      lam_combo[count2] = lambda[index[count2]][count2];
                     diff_sum_temp += fabs(lam_combo[count2] - prop_lam[count2]);
   }
           if (diff_sum_temp > diff_sum_min){
                     if (diff_sum_temp < diff_sum){
                              for ( count2 = 0; count2 <= n ; count2++){
                                       lam_sol[count2] = lam_combo[count2];
                                       diff_sum = diff_sum_temp;
                              }
                     }
           }
           }
           diff_sum_min = diff_sum;
           diff_sum = 20;

//implementation of sequentially increasing lambda constraint
   int flag = 1;

   for (count6 = 1; count6 <= n; count6++){
      if (lam_sol[count6] < lam_sol[(count6 - 1)])
         flag = 0;
   }
 /*  if (flag == 0){
      cout << "The lambda set [ ";
      for (count6 = 0; count6 <= n; count6++)
      {
      cout << setw(10) << lam_sol[count6];
      }
      cout << " ] is infeasible!" << endl << endl;
   }*/
   if (flag == 1){
      iterate++;

//assignment of dummy variables
      for (j = 0; j < mrow; j++){
         for (i = 0; i < mcol; i++){
            if (arr_time[j][i] <= (cs + (lt * (ki[j][i]-1)) + (h * lam_sol[(ki[j][i]) - 1])))
```

```
            xdummy[j][i] = 1;
         else xdummy[j][i] = 0;
         if (arr_time[j][i] <= (cs + (lt * (ki[j][i]-1)) + (h * lam_sol[(ki[j][i] - 1)])))
            ydummy[j][i] = 0;
         else if (arr_time[j][i] > (cs + (h * lam_sol[(ki[j][i])])))
            ydummy[j][i] = 0;
         else ydummy[j][i] = 1;
         if (arr_time == 0)
            xdummy[j][i] = 0, ydummy[j][i] = 0;
      }
   }
```

//calculation of delay via objective function

```
      total_del = 0;
                              total_stop_del = 0;
                              co_del = 0;
      for (j = 0; j < arr; j++)
      {
         for (i = 0; i < m; i++)
         {
            stop_del[j][i] = (1 - ydummy[j][i]) * ((cs + (xdummy[j][i]*lam_sol[(ki[j][i] - 1)]*h) + ((1 -
xdummy[j][i])*(h+(n*lt))) - arr_time[j][i]) + ((2*xdummy[j][i]) + xdummy[j+1][i] + xdummy[j+2][i]));
            total_stop_del += stop_del[j][i];
         }
      }
                              for (j=0; j<=(n-3); j++){
                                    co_del += (cntr[j] * (cs + (lam_sol[j+1]*h))) - sum_at[j];
                              }
                              total_del = total_stop_del + co_del;
```

//display of dummy variables and delay calculations for each lambda solution set

```
/*          cout << "For lambdas [";
      for (count6 = 0; count6 <= n; count6++)
      {
         cout << setw(10) << lam_sol[count6];
      }
      cout << "   ] :" << endl;

      cout << "The proportional ranking value is: ";
                              prop_rank = 0;
      for (count6 = 0; count6 <= n; count6++)
      {
         prop_rank += fabs(lam_sol[count6] - prop_lam[count6]);
      }
      cout << prop_rank << endl;

      cout << "Here are the x dummy variables for each vehicle:" << endl;
      for (j = 0; j < arr; j++)
      {
         for (i = 0; i < m; i++)
         {
         cout << setw(8) << xdummy[j][i];
         }
         cout << endl;
```

```
        }
        cout << endl;
        cout << "Here are the y dummy variables for each vehicle:" << endl;
        for (j = 0; j < arr; j++)
        {
           for (i = 0; i < m; i++)
           {
           cout << setw(8) << ydummy[j][i];
           }
           cout << endl;
        }
        cout << "Here are the stopped delays for each vehicle:\n";
        for (j = 0; j < arr; j++)
        {
           for (i = 0; i < m; i++)
           {
              cout << setw(8) << stop_del[j][i];
           }
           cout << endl;
        }
        cout << "The total stopped delay is: " << total_del << endl;
        cout << endl;
*/
//storage of optimal solution
        if (total_del < delay){
           delay = total_del;
           iter_opt = iterate;
           for (i = 0; i <= n; i++){
              lam_out[i] = lam_sol[i];
           }
          exp_del = 0;
           for (i = 0; i<=(n-3); i++){
              cntr_temp[i] = 0;
          sum_at_temp[i]=0;
           }
                              for (j = 0; j < arr; j++){
                              for (i = 0; i < m; i++){
                                    if (xdummy[j][i] == 1){
                                          if (ki[j][i] == 2){
                                                exp_del += stop_del[j][i];
                                          }
                                          if (ki[j][i] >= 3){
                                          if (arr_time[j][i] > (lam_sol[1]*h)){
                                                cntr_temp[(ki[j][i]-3)]++;
                                                sum_at_temp[(ki[j][i]-3)] += arr_time[j][i];
                                          }
                                          }
                                    }
                                          }
                              }
                              }
        //cout << endl;
    }
```

```
    }
        exp_del += (cntr[1] * (cs + (lam_sol[1]*h))) - sum_at[1];
        for (i=0; i<=(n-3); i++){
                cntr[i] = cntr_temp[i];
                sum_at[i] = sum_at_temp[i];
                if (i==(n-3)){
                        cntr_temp[i]=0;
                        sum_at_temp[i]=0;
                }
                else{
                        cntr_temp[i]=cntr_temp[i+1];
                        sum_at_temp[i]=sum_at_temp[i+1];
                }
        }
```

//display of final solution for phase for given iteration

```
//    cout << "The run time is now: " << rt << " seconds.\n";
    cout << "The minimum stopped delay is: " << delay << " vehicle-seconds.\n";
    cout << "It occurs at iteration " << iter_opt << " of the algorithm.\n";
    cout << "It occurs at the lambda set: [";
    for (count6 = 0; count6 <= n; count6++) {
       cout << setw(10) << lam_out[count6];
    }
    cout << " ]" << endl;
        cout << "Therefore, phase " << phase << " should have a green time length of " << (lam_out[1] *
(h+(n*lt))) << " seconds.\n";
        cout << "It should end at " << (cs + (lam_out[1] * (h+(n*lt)))) << " seconds.\n";
    cout << "There were " << NC << " possible combinations of lambdas, of which " << iterate << " were
feasible.\n";
    cout << endl;
        cum_del += exp_del;
        cout << "The experienced stopped delay for this phase is: " << exp_del << " vehicle-seconds.\n";
        cout << "The cumulative experienced delay for this run is: " << cum_del << " vehicle-seconds.\n";
```

//update

```
    pe = cs + (lam_out[1] * h);
```

//reinitialization

```
//    cout << "The run time has expired for this phase.\n";
    cout << "Continue to the next phase? (1 for yes, 2 for no)\n";
    cin >> no;
    cs = cs + (lam_out[1] * (h+(n*lt))) ;
    rt = cs;
        if (phase == n)
                phase = 1;
        else phase = phase + 1;
    for (count6 = 0; count6 < n; count6++){
                k[count6] = (k[count6] - 1);
                if (k[count6] == 0)
                        k[count6] = n;
                }
}
```

```
    cout << "Thanks for using PAD optimization today!\n";
    return 0;
}
```

Code for Metaheuristic Search

```
/*#include <stdlib.h>
//#include "stdafx.h"
#include <iostream>
using std::ios;
#include <fstream>
using std::ifstream;
#include <iomanip>
#include <string>
#include < math.h>
#include <new>
using namespace std;

using std::string;
#using <mscorlib.dll>
*/

//Opening of header files to be inserted into program
#include <iostream.h>
#include < iomanip.h>
#include <fstream.h >
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <new.h>

int main()
{
//inventory of introduced variables
   const int mrow = 1800;
   const int mcol = 8;
   int arr, i, j, m, no, count, count2, count3, count6, position, iterate, iter_opt, manin, lt, phase, count_iter, tt,
tabu_ind, leap;
   double veh_count[mcol], prop_lam[mcol], at[mrow][mcol], arr_time[mrow][mcol], lam[mrow][mcol],
stop_del[mrow][mcol], lambda[mrow][mcol], tabu_char[mcol], diff_sum[mrow], lam_sol[mcol],
lam_out[mcol], lam_combo[mrow][mcol], sum_at[mcol], sum_at_temp[mcol], q_del[mrow][mcol],
temp2[mcol];
   int xdummy[mrow][mcol], ydummy[mrow][mcol], ki[mrow][mcol], k[mcol],  cntr[mcol],
cntr_temp[mcol], tabu_index[mcol];
   double h, l_inc, delay, total_del, total_stop_del, exp_del, cum_del, cs, rt, ts, pe, n, sim_end, lam_max,
prop_rank, co_del, temp1;

   ifstream Input;


/*    for (i=0;i<mrow;i++){
       for (j=0;j<mcol;j++){
          cout<<stop_del[i][j]<<"\t";
       }
       cout<<"\n";
    }
*/
```

```
//initialization
   cs = 0;
   rt = 0;
            sim_end = 3600;
            phase = 1;
   no = 1;
   ts = 0;
            cum_del = 0;
            tt = 4;
   for (i=0; i < mcol; i++){
      k[i] = 0;
                        cntr[i] = 0;
                        sum_at[i] = 0;
                        cntr_temp[i] = 0;
                        sum_at_temp[i] = 0;
   }

//user definition of variables
   cout << "Welcome to the PAD program for adaptive traffic signal control! \n";
            cout << "Please enter all time values in seconds.\n";
   cout << "For how long would you like the simulation to be able to run?\n";
            cin >> sim_end;
   cout << "What would you like the horizon length (assumed cycle length) to be?\n";
   cin >> h;
   cout << "What would you like the minimum discernible lambda increment to be?\n";
   cin >> l_inc;
   cout << "What would you like the lost (all-red) time to be for each phase?\n";
   cin >> lt;
/*    cout << "What time step would you like the program to reiterate over (typically 0.5 to 5 seconds)?\n";
   cin >> ts;*/
   cout << "How many approaches do you have to the intersection?\n";
   cin >> m;
   cout << "How many phases would you like to use for the cycle?\n";
   cin >> n;
   cout << endl;
            cout << "The assumed cycle length will be adjusted for lost time to " << h + (n * lt) << "
seconds.\n";
   cout << "Would you like to manually input arrival times [type '1'] or get them from the 'arr_time.txt' file
[type '2']?\n";
   cin  >> manin;
   cout << endl;
   for (count6=0; count6<m; count6++){
   cout << "What phase is approach " << count6 + 1 << " being assigned to? \n";
   cin >> k[count6];
   }


//phase-to-phase process condition (user-defined)
while (no == 1){

   for (j=0; j < mrow; j++){
      for (i=0; i < mcol; i++){
```

```
            arr_time[j][i] = 0;
                                    lam[j][i] = 0;
                                    lambda[j][i] = 0;
                                    ki[j][i] = 0;
        }
    }

    for (i=0; i < mcol; i++){
        lam_sol[i] = 0;
                        tabu_char[i] = 0;
                        tabu_index[i] = 0;
    }

    pe = cs + h;

    tabu_ind = n;
    arr = 0;
    count6 = 0;
    iterate = 0;
              rt = rt + ts;
              cout << endl;
    cout << "The cycle start time is now " << cs << " and the nominal cycle end time is " << (cs+(h+(n*lt)))
<< ".\n";

//arrival time setup for manual input
    if (manin == 1){
    cout << "Please enter arrival times within the range of the cycle start and end times.\n";
    while (count6 < m){
                        cout << "For approach " << count6 + 1 << ": \n";
        cout << "How many vehicles are arriving? \n";
        cin >> j;
        cout << "Enter the arrival time of each vehicle: \n";
            for (count = 0; count < j; count++){
                cout << "Vehicle " << (count + 1) << ": ";
                cin >> arr_time[count][count6];
                ki[count][count6] = k[count6];
            }
        cout << endl;
        count6++;
        if (j > arr)
            arr = j;
    }
    cout << endl;
    }

//arrival time setup for file input
    if (manin == 2){

        Input.open("arr_time.txt");

        for (j=0;j<mrow;j++){
            for (i=0;i<mcol;i++){
                Input>>at[j][i];
```

```
            }
        }
        Input.close();
/*      for (j=0;j<mrow;j++){
          for (i=0;i<mcol;i++){
             cout<<setw(8)<<at[j][i];
          }
          cout << endl;
        }*/
        count = 0;
        count2 = 0;
        for (i=0;i<mcol;i++){
           for (j=0;j<mrow;j++){
              if (at[j][i] > (cs)){
                 if (at[j][i] <= (cs + (n*lt) + h)){
                    arr_time[count][count2] = at[j][i];
                    ki[count][count2] = k[count2];
                    count++;
                 }
              }
           }
        count=0;
        count2++;
        }

        for (i=0;i<mcol;i++){
           count6 = 0;
           for (j=0;j<mrow;j++){
              if (arr_time[j][i] > 0){
                 count6++;
              }
           if (count6 > arr)
              arr = count6;
           }
        }
     }

//lambda setup based on vehicle arrival times
     for (j = 0; j < arr; j++){
        for (i = 0; i < m; i++){
        if (arr_time[j][i] > 0)
           lam[j][i] = ((arr_time[j][i] - cs) / (h + (n*lt))) + l_inc;
        }
     }

//arrangement of lambda solution space based on phasing
     count2 = 0;
     lam_max = 0;
     for (count = 1; count <= n; count++){
        for (j = 0; j < arr; j++){
           for (i = 0; i < m; i++){
              if (lam[j][i] > 0)
                 if (ki[j][i] == count)
```

```
                lambda[count2][count] = lam[j][i];
            if (ki[j][i] == count)
                count2++;
            if (count2 > lam_max)
                                                    lam_max = count2;
        }
      }
      count2 = 0;
   }

//display of independent variables for confirmation
   cout << "Here is the arrival time matrix for this cycle:" << endl;
   for (j = 0; j < arr; j++){
      for (i = 0; i < m; i++){
         cout << setw(8) << arr_time[j][i];
      }
      cout << endl;
   }
   cout << endl;

   cout << "Here are the phases for each vehicle:" << endl;
   for (j = 0; j < arr; j++){
      for (i = 0; i < m; i++){
         cout << setw(8) << ki[j][i];
      }
      cout << endl;
   }
   cout << endl;

   cout << "Here is the lambda matrix to be used:" << endl;
   for (j = 0; j <= lam_max; j++){
      for (i = 0; i <= n; i++){
         cout << setw(10) << lambda[j][i];
      }
      cout << endl;
   }
   cout << endl;

   delay = 1000000;

//solution selection for exhaustive enumeration
/*    int *index;
   index=new int [n+1];

   for (i = 0; i <= n; i++){
      index[i] = 0;
   }

   double NC = pow((lam_max+1), n);

   for ( count = 1; count <= NC; count++ ){

   if ( count == 1 ){
```

```
      for ( count2 = 1; count2 <= n; count2++ )
         index[count] = 0;
      }
   else {
      position = n;
      while ( index[position] == lam_max )
         position--;
      index[position]++;
      for ( count3 = position+1; count3 <= n; count3++ )
         index[count3] = 0;
           }

           for ( count2 = 0; count2 <= n ; count2++){
      lam_sol[count2] = lambda[index[count2]][count2];
   }

*/

//initial solution selection via proportional heuristic
   int *index;
   index=new int [n+1];

   for (i = 0; i <= n; i++){
      index[i] = 0;
   }

   double NC = pow((lam_max+1), n);

           for (i = 0; i <= mrow; i++){
      diff_sum[i] = 0;
   }

           count_iter = 0;

   for ( count = 1; count <= NC; count++ ){

   if ( count == 1 ){
      for ( count2 = 1; count2 <= n; count2++ )
         index[count2] = 0;
      }

   else {
      position = n;
      while ( index[position] == lam_max )
         position--;
      index[position]++;
      for ( count3 = position+1; count3 <= n; count3++ )
         index[count3] = 0;
           }

           double total_veh=0;

           for (i=0; i<=n; i++){
```

```
                veh_count[i] = 0;
                prop_lam[i]=0;
        }

        for (j=0; j<=lam_max; j++){
                for (i=0; i<=n; i++){
                        if (lambda[j][i] > 0){
                                veh_count[i]++;
                                total_veh++;
                        }
                }
        }

for (i = 0; i <= n; i++){
   prop_lam[i] = (veh_count[i])/total_veh;
}

        for ( count2 = 0; count2 <= n ; count2++){
                lam_sol[count2] = lambda[index[count2]][count2];
        }

        int flag = 1;
        for (count6 = 1; count6 <= n; count6++){
   if (lam_sol[count6] < lam_sol[(count6 - 1)])
      flag = 0;
}

        if (flag == 1){
                for (count6 = 0; count6 <= n; count6++){
                        diff_sum[count_iter] += fabs(lam_sol[count6] - prop_lam[count6]);
                        lam_combo[count_iter][count6] = lam_sol[count6];

                }
                count_iter++;
        }
        }

        for (count=0; count<count_iter; count++){
                if (diff_sum[count] > diff_sum[count+1]){
                        temp1 = diff_sum[count];
                        for (count2=0; count2<=n; count2++){
                                temp2[count2] = lam_combo[count][count2];
                        }
                        diff_sum[count]=diff_sum[count+1];
                        for (count2=0; count2<=n; count2++){
                                lam_combo[count][count2] = lam_combo[count+1][count2];
                        }
                        diff_sum[count+1]=temp1;
                        for (count2=0; count2<=n; count2++){
                                lam_combo[count+1][count2] = temp2[count2];
                        }
                }
        }
```

```
        if (count_iter < 500)
                count3 = (count_iter - 1);
        else count3 = 500;

        leap = 0;

//metaheuristic selection
for (count=0; count<count3; count++){

        if (leap==(count3-1))
                leap=0;
        else leap++;

        for (count2 = 1; count2 <= n ; count2++){
                lam_sol[count2] = lam_combo[leap][count2];
        }

        if (count > 0){
                for (j=0;j<=tt;j++){
                        if(lam_sol[(tabu_index[j])] = tabu_char[j]){
                                leap++;
                                for (count2 = 1; count2 <= n ; count2++){
                                        lam_sol[count2] = lam_combo[leap][count2];
                                }
                        }
                }
        }

//assignment of dummy variables
      for (j = 0; j < mrow; j++){
         for (i = 0; i < mcol; i++){
            if (arr_time[j][i] <= (cs + ((h+(n*lt)) * lam_sol[(ki[j][i]) - 1])))
               xdummy[j][i] = 1;
            else xdummy[j][i] = 0;
            if (arr_time[j][i] <= (cs + ((h+(n*lt)) * lam_sol[(ki[j][i] - 1)])))
               ydummy[j][i] = 0;
            else if (arr_time[j][i] > (cs + ((h+(n*lt)) * lam_sol[(ki[j][i])])))
               ydummy[j][i] = 0;
            else ydummy[j][i] = 1;
            if (arr_time == 0)
               xdummy[j][i] = 0, ydummy[j][i] = 0;
         }
       }

//calculation of delay via objective function
         total_del = 0;
                                total_stop_del = 0;
                                co_del = 0;
         for (j = 0; j < arr; j++)
         {
            for (i = 0; i < m; i++)
            {
```

```
                stop_del[j][i] = (1 - ydummy[j][i]) * ((cs + (xdummy[j][i]*lam_sol[(ki[j][i] - 1)]*(h+(n*lt))) +
((1 - xdummy[j][i])*(h+(n*lt))) - arr_time[j][i]));
                                                        q_del[j][i] =  (1 - ydummy[j][i]) * ((2*xdummy[j][i]) +
xdummy[j+1][i] + xdummy[j+2][i]);
                total_stop_del += (stop_del[j][i] + q_del[j][i]);
             }
          }
                                  for (j=0; j<=(n-3); j++){
                                          co_del += (cntr[j] * (cs + (lam_sol[j+1]*h))) - sum_at[j] + (2 * cntr[j]);
                                  }
                                  total_del = total_stop_del + co_del;

//display of dummy variables and delay calculations for each lambda solution set
/*          cout << "For lambdas [";
          for (count6 = 0; count6 <= n; count6++)
          {
             cout << setw(10) << lam_sol[count6];
          }
          cout << "   ] :" << endl;

          cout << "The proportional ranking value is: ";
                                  prop_rank = 0;
          for (count6 = 0; count6 <= n; count6++)
          {
             prop_rank += fabs(lam_sol[count6] - prop_lam[count6]);
          }
          cout << prop_rank << endl;

          cout << "Here are the x dummy variables for each vehicle:" << endl;
          for (j = 0; j < arr; j++)
          {
             for (i = 0; i < m; i++)
             {
             cout << setw(8) << xdummy[j][i];
             }
             cout << endl;
          }
          cout << endl;
          cout << "Here are the y dummy variables for each vehicle:" << endl;
          for (j = 0; j < arr; j++)
          {
             for (i = 0; i < m; i++)
             {
             cout << setw(8) << ydummy[j][i];
             }
             cout << endl;
          }
          cout << "Here are the stopped delays for each vehicle:\n";
          for (j = 0; j < arr; j++)
          {
             for (i = 0; i < m; i++)
             {
                cout << setw(8) << stop_del[j][i];
```

```
            }
            cout << endl;
        }
        cout << "The total stopped delay is: " << total_del << endl;
        cout << endl;
*/
//storage of optimal solution
        if (total_del < delay){
            delay = total_del;
            iter_opt = (count+1);
            for (i = 0; i <= n; i++){
                lam_out[i] = lam_sol[i];
            }
                                    exp_del = 0;
            for (i = 0; i<=(n-3); i++){
                cntr_temp[i] = 0;
                                            sum_at_temp[i]=0;
            }
                                    for (j = 0; j < arr; j++){
                                            for (i = 0; i < m; i++){
                                                    if (xdummy[j][i] == 1){
                                                            if (ki[j][i] == 2){
                                                                    exp_del += stop_del[j][i];
                                                            }
                                                            if (ki[j][i] >= 3){
                                                                    if (arr_time[j][i] <
(lam_sol[1]*(h+(n*lt)))){
                                                                            cntr_temp[(ki[j][i]-3)]++;
                                                                            sum_at_temp[(ki[j][i]-3)]
+= arr_time[j][i];
                                                                    }
                                                            }
                                                    }
                                            }
                                    }
                            }
//reduction of tabu tenure
                            for (j=0; j<=tt; j++){
                                    if (j==tt){
                                            tabu_char[j]=0;
                                            tabu_index[j]=0;
                                    }
                                    else{
                                            tabu_char[j]=tabu_char[j+1];
                                            tabu_index[j]=tabu_index[j+1];
                                    }
                            }

//placement of tabu status
                            if (total_del > delay){
                                    tabu_char[0]=lam_sol[tabu_ind];
                                    tabu_index[0]=tabu_ind;
                                    if (tabu_ind == 1)
```

```
                                        tabu_ind = n;
                                    else tabu_ind--;
                                }

        //cout << endl;
    }


//update of carrover delay arrays
            exp_del += (cntr[1] * (cs + (lam_sol[1]*h))) - sum_at[1];
            for (i=0; i<=(n-3); i++){
                    cntr[i] = cntr_temp[i];
                    sum_at[i] = sum_at_temp[i];
                    if (i==(n-3)){
                            cntr_temp[i]=0;
                            sum_at_temp[i]=0;
                    }
                    else{
                            cntr_temp[i]=cntr_temp[i+1];
                            sum_at_temp[i]=sum_at_temp[i+1];
                    }
            }


//display of final solution for phase for given iteration
//    cout << "The run time is now: " << rt << " seconds.\n";
    cout << "The minimum stopped delay is: " << delay << " vehicle-seconds.\n";
    cout << "It occurs at iteration " << iter_opt << " of the algorithm.\n";
    cout << "It occurs at the lambda set: [";
    for (count6 = 0; count6 <= n; count6++) {
        cout << setw(10) << lam_out[count6];
    }
    cout << " ]" << endl;
            cout << "Therefore, phase " << phase << " should have a green time length of " << (lam_out[1] *
(h+(n*lt))) << " seconds.\n";
            cout << "It should end at " << (cs + (lam_out[1] * (h+(n*lt)))) << " seconds.\n";
    cout << "There were " << NC << " possible combinations of lambdas, of which " << (count_iter-1) << "
were feasible.\n";
    cout << endl;
            cum_del += exp_del;
            cout << "The experienced stopped delay for this phase is: " << exp_del << " vehicle-seconds.\n";
            cout << "The cumulative experienced delay for this run is: " << cum_del << " vehicle-seconds.\n";

//update
    pe = cs + (lam_out[1] * h);


//reinitialization
//    cout << "The run time has expired for this phase.\n";

    cout << "Continue to the next phase? (1 for yes, 2 for no)\n";
```

```
   cin >> no;

   cs = cs + (lam_out[1] * (h+(n*lt))) + lt;
   rt = cs;
          if (phase == n)
                 phase = 1;
          else phase = phase + 1;
   for (count6 = 0; count6 <= n; count6++){
                 k[count6] = (k[count6] - 1);
                 if (k[count6] == 0)
                        k[count6] = n;
                 }
//
   if (pe >= sim_end){
                 no = 2;
                 cout << "The simulation end time has been reached!\n";
          }
}

   cout << "Thanks for using PAD optimization today!\n";
   return 0;
}
```

Appendix 2: Coding a Run-Time Extension (RTE) for TSIS/CORSIM

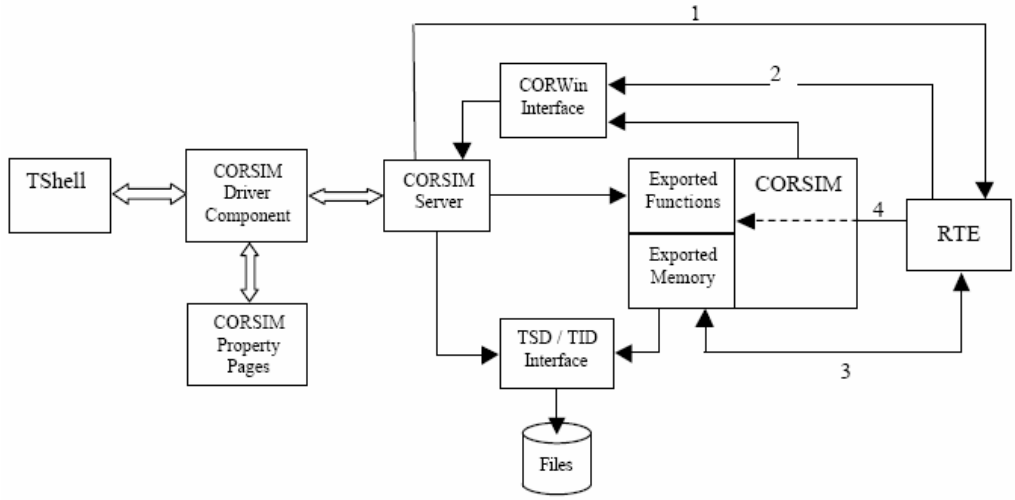

Figure A2.1: "The Run-Time Extension Interface" (RTE Developer's Guide, ITT 2003)

Figure A2.1 shows the interaction between TSIS and a potential RTE. Link 1 allows CORSIM to initialize and exit the RTE as it is initialized or exited. Link 2 allows the RTE to communicate with the CORSIM server, controlling the display to messages in CORSIM relating to the status of the RTE, e.g. initialization errors, run errors, status messages, etc. In creating the RTE for adaptive control, the code pertaining to these two links would be nearly identical to any other RTE (except, possibly, for specific messages dealing with the adaptive algorithm that could be displayed in the code dealing with link 2, such as constraint violations and/or infeasibilities).

Link 4 allows the RTE to "provide execution control, path-based vehicle control and data access," according to Section 3.4 of the RTE Developer's Guide (ITT 2003); it is recommended that the RTE not implement code pertaining to this link, with the

exception of the “abortcorsim” function, which can abort a simulation run based on criteria specified by the developer in the RTE. It may also be necessary or desirable to access the WRITE_OUTPUTFILE command to incorporate results from the execution of the algorithm in the RTE into the CORSIM output file.

Manipulation of the code dealing with link 3 is the key to creating a run-time extension specific to the adaptive control scheme we have created. Link 3 allows information to be exchanged between CORSIM and the RTE, either for information purposes or for direct use in the execution of functions in the respective codes. This is different from the code that controls the interaction in link 4 because, whereas link 4 code actually controls CORSIM functions as they are carried out, link 3 functions generate arrays of data which are then imported into or exported out of CORSIM. These arrays are updated as the CORSIM and RTE simulations are carried out, and accessed as needed by the two simulations.

The CORSIM Data dictionary lists the data arrays compiled and accessed by CORSIM. Using code pertaining to link 3, as described in Section 3.3 of the RTE Developer’s Guide, we can obtain the data in these arrays and manipulate it as desired. Among the arrays we may wish to export from CORSIM for use in the adaptive control algorithm in the RTE are the following (common refers to the common FORTRAN block to which CORSIM assigns this array):

use	array name	common	component data
output from CORSIM	IDTIME(DT)	SIN341	Elapsed timed since beginning of the current time step, in tenths-of-a-second, when the first actuation was recorded.
	LASTD(DT)	SIN342	Elapsed timed since beginning of the current time step, in tenths-of-a-second, when the last actuation was recorded.
	LVACT(DT)	SIN367	Last vehicle ID that activated the detector.
input to CORSIM	SDCODE(IL)	(n/a)	Signal codes of current, active, signal interval controlling traffic on this link, fetched from XINT1 or XINT2. 1 Signal code for right turn vehicles 2 Signal code for through vehicles 3 Signal code for diagonal (left or right) vehicles 4-5 Signal code for left turn vehicles Where Signal Code is defined as follows: 0 GO, permitted and protected 1 NO GO, not permitted 2 COND, GO, permitted but unprotected (left turners only)

The output arrays are what are referred to in the Developer's Guide as statically allocated arrays, where the dimension of the array is fixed during run time. Section 3.3.2 of the guide outlines the coding to access an array of this type. For instance, to export the array LASTD for a detector labeled ART001, we would use the following code:

```
#define ART001
DLL_IMPORT struct{ int LASTD[ART001]; } SIN342;
#define dtmod SIN342.LASTD
```

The SDCODE array is slightly different; it is not assigned to a common block, but is instead part of a dynamically allocated array in the NETSIM Link Database. As Section 3.3.3 of the Developer's Guide points out, this array would be called with the following code:

```
DLL_IMPORT int* NETSIM_LINKS_mp_SDCODE;
#define dwnod NETSIM_LINKS_mp_SDCODE
```

All of these arrays contain data that is extraneous to the adaptive algorithm. Section 2.1 of the RTE Developer's Guide, provides the coding necessary to prepare the data to be used by the RTE. These steps vary for statically and dynamically allocated arrays; statically allocated arrays can be "trimmed" by the code to reach the necessary data, while dynamically allocated arrays must be searched.

There is also the issue of needing to build the three output arrays from CORSIM listed above into arrays that can be utilized by the adaptive control scheme, which requires inputs of vehicle number, and vehicle arrival (detector actuation) time for each link. Furthermore, this data will not be used by the adaptive algorithm directly, but by a predictive algorithm which will output the above data for a set time horizon after the current time step of the simulation (Also, in this regard, CORSIM clearly works in time steps of 0.1-second-length, so the resolution of the adaptive control scheme will be set to this level.) This should be possible by creating code which will create this new array and update it with each time step in the predictive algorithm, and then using code which will access the data in the new array in the adaptive algorithm.

The pseudo-code for this procedure will be akin to the one in Figure 8.1, with the following modifications:

- Instead of the detection process pseudo-code, detector data will be extracted from CORSIM using the following pseudo-code:

  [**do** for all CORSIM output arrays]
  [access CORSIM output array using code in Section 3.3.2 of Guide]
  ["trim" data in array using code in Section 2.1.2 of Guide]

[set data to position in existing $\mathbf{t}_{ij}$ array]

- The output step will send the optimum `1` to CORSIM instead of the signal controller

This pseudo-code will likely be required to be altered for an expansion of the network from one intersection, but the iterative process inherent to it will remain. As it is written, it should be functional for multiple phasing schemes on a single intersection.

Although the coding can be somewhat involved for the RTE, the TShell tool included in version 5.1 of TSIS allows easy configuration of a compiled RTE for use in a TSIS simulation. Some of the settings that may be made here include setup of input and output files, message settings for link 2 coding, data pattern settings (for vehicle, traffic, and headway, to facilitate comparison over multiple runs with differing conditions), initialization, main, and exit function organization, and even use of multiple RTE's.

Coding for the RTE and the algorithm, as well as the setup of the CORSIM simulation, must be made in sync. It is likely that a separate code, as a modification of the generic adaptive control scheme code, will be developed for use specifically with the Run-Time Extension. The sample code included with the RTE Toolkit downloaded from FHWA's website, will provide additional guidance in this regard and well as much of the RTE coding, as it was developed in order to test the RT-TRACS adaptive control scheme in CORSIM.

Appendix 3: Data and Excel Implementation for Experiment #1

Experiment #1

Run #1 arrival times

approach	1.00	2.00	3.00	4.00
phase	2.00	3.00	1.00	1.00
gap	25.00	30.00	35.00	40.00
1.00	10.00	15.00	35.00	30.00
2.00	35.00	45.00	70.00	70.00
3.00	60.00	75.00	105.00	110.00
4.00	85.00	105.00	140.00	150.00
5.00	110.00	135.00	175.00	190.00
6.00	135.00	165.00	210.00	230.00
7.00	160.00	195.00	245.00	270.00
8.00	185.00	225.00	280.00	310.00
9.00	210.00	255.00	315.00	350.00

Run #2 arrival times

approach	1.00	2.00	3.00	4.00
phase	2.00	3.00	1.00	1.00
gap	35.00	30.00	30.00	25.00
1.00	5.00	10.00	15.00	20.00
2.00	30.00	30.00	45.00	45.00
3.00	55.00	50.00	75.00	70.00
4.00	80.00	70.00	105.00	95.00
5.00	105.00	90.00	135.00	120.00
6.00	130.00	110.00	165.00	145.00
7.00	155.00	130.00	195.00	170.00
8.00	180.00	150.00	225.00	195.00
9.00	205.00	170.00	255.00	220.00

Run #3 random numbers

approach	1.00	2.00	3.00	4.00
phase	2.00	3.00	1.00	1.00
veh/hr	200.00	100.00	50.00	50.00
1.00	0.25	0.40	0.43	0.52
2.00	0.63	0.46	0.25	0.07
3.00	0.78	0.08	0.32	0.75
4.00	0.60	0.02	0.57	0.27
5.00	0.48	0.91	0.50	0.68
6.00	0.76	0.01	0.83	0.41
7.00	0.62	0.45	0.18	0.12
8.00	0.45	0.57	0.65	0.45
9.00	0.54	0.19	0.61	0.71

10.00	0.56	0.88	0.80	0.91
11.00	0.24	0.35	0.71	0.76
12.00	0.54	0.15	0.14	0.06
13.00	0.06	0.58	0.10	0.36

Run #3 interarrival times

approach	1.00	2.00	3.00	4.00
phase	2.00	3.00	1.00	1.00
veh/hr	150.00	100.00	50.00	50.00
1.00	7.06	18.18	40.11	53.38
2.00	24.01	22.20	20.29	5.41
3.00	36.87	3.01	28.03	99.45
4	21.7912	0.654821	61.16249	23.0293
5	15.56366	85.11157	49.67482	82.23818
6	34.17141	0.471027	125.8021	38.3637
7	23.3902	21.25304	14.16541	8.912984
8	14.46805	30.09228	74.83305	43.54925
9	18.66611	7.585258	67.04416	89.18364
10	19.59767	75.31643	114.8045	174.2816
11	6.654251	15.3316	89.92802	101.8233
12	18.89063	5.819528	10.55346	4.549645
13	1.498397	31.20965	7.729778	32.45153

Run #3 arrival times

approach	1	2	3	4
phase	2	3	1	1
veh/hr	150	100	50	50
1	7.059709	18.17516	40.10526	53.37972
2	31.07266	40.3731	60.39987	58.78625
3	67.93836	43.38164	88.42574	158.2335
4	89.72955	45.38164	149.5882	181.2628
5	105.2932	130.4932	199.2631	263.501
6	139.4646	132.4932	325.0651	301.8647
7	162.8548	153.7462	339.2305	310.7777
8	177.3229	183.8385	414.0636	354.3269
9	195.989	191.4238	481.1078	443.5106
10	215.5867	266.7402	595.9122	617.7922
11	222.2409	282.0718	685.8403	719.6155
12	241.1315	287.8913	696.3937	724.1651
13	243.1315	319.101	704.1235	756.6167

Experiment #1

Run #1 Iteration #1 lambda set			start	0	end	63			
iteration 1									
phase	1	2	3						
1	0.557	0.160	0.239						
2	0.477	0.557	0.715						
3	0.000	0.953	0.000						
4	0.000	0.000	0.000						
carryover	sum_time	phase	cntr						
	0	0	0						

Run #1 Iteration #1									
Delay table			0.000	35.091	35.091	45.045			
combination 6			0.000	**0.557**	**0.557**	**0.715**		carryover	0.000
ID	tij	xij	ki	bg delay	ag delay	yij	delay	q_del	
delay 11	10	1	2	25.091	53	0	25.091	3	
delay 12	35	1	2	0.091	28	0	0.091	2	
delay 13	60	0	2	-24.909	3	0	3	0	
delay 21	15	1	3	20.091	48	0	20.091	2	
delay 22	45	0	3	9.909	18	1	0	0	
delay 31	35	0	1	-35.000	28	1	0	0	
delay 41	30	0	1	-30.000	33	1	0	0	
							48.273	7	
							sum	55.273	
			0	30.051	35.091	45.045			
combination 22			0	**0.477**	**0.557**	**0.715**		carryover	0.000
ID	tij	xij	ki	bg delay	ag delay	yij	delay	q_del	
delay 11	10	1	2	20.051	53	0	20.051	2	
delay 12	35	0	2	-4.949	28	1	0	0	
delay 13	60	0	2	-29.949	3	0	3	0	
delay 21	15	1	3	20.091	48	0	20.091	2	
delay 22	45	0	3	9.909	18	1	0	0	
delay 31	35	0	1	-35.000	28	0	28	0	
delay 41	30	0	1	-30.000	33	1	0	0	
							71.142	4	
							sum	75.142	

			0	0	10.08	15.057			
combination 33			0	**0.000**	**0.160**	**0.239**		carryover	0.000
ID	tij	xij		ki	bg delay	ag delay	yij	delay	q_del
delay 11	10	0		2	-10	53	1	0	0
delay 12	35	0		2	-35	28	0	28	0
delay 13	60	0		2	-60	3	0	3	0
delay 21	15	0		3	-4.92	48	1	0	0
delay 22	45	0		3	34.92	18	0	18	0
delay 31	35	0		1	-35.000	28	0	28	0
delay 41	30	0		1	-30.000	33	0	33	0
								110	0
								sum	110.000

			0	0	10.08	45.045			
combination 34			0	**0.000**	**0.160**	**0.715**		carryover	0.000
ID	tij	xij		ki	bg delay	ag delay	yij	delay	q_del
delay 11	10	0		2	-10	53	1	0	0
delay 12	35	0		2	-35	28	0	28	0
delay 13	60	0		2	-60	3	0	3	0
delay 21	15	0		3	-4.92	48	1	0	0
delay 22	45	0		3	34.92	18	1	0	0
delay 31	35	0		1	-35.000	28	0	28	0
delay 41	30	0		1	-30.000	33	0	33	0
								92	0
								sum	92.000

			0	0	35.091	45.045			
combination 38			0	**0.000**	**0.557**	**0.715**		carryover	0.000
ID	tij	xij		ki	bg delay	ag delay	yij	delay	q_del
delay 11	10	0		2	-10	53	1	0	0
delay 12	35	0		2	-35	28	1	0	0
delay 13	60	0		2	-60	3	0	3	0
delay 21	15	1		3	20.091	48	0	20.091	2
delay 22	45	0		3	9.909	18	1	0	0
delay 31	35	0		1	-35.000	28	0	28	0
delay 41	30	0		1	-30.000	33	0	33	0
								84.091	2
								sum	86.091

			0.000		0	0	15.063			
combination 45			0.000		**0.000**	**0.000**	**0.239**		carryover	0.000
ID	tij		xij		ki	bg delay	ag delay	yij	delay	q_del
delay 11		10	0		2	-10	53	0	53	0
delay 12		35	0		2	-35	28	0	28	0
delay 13		60	0		2	-60	3	0	3	0
delay 21		15	0		3	-15	48	1	0	0
delay 22		45	0		3	45	18	0	18	0
delay 31		35	0		1	-35.000	28	0	28	0
delay 41		30	0		1	-30.000	33	0	33	0
									163	0
									sum	163.000
			0		0	0	45.063			
combination 46			0		**0.000**	**0.000**	**0.715**		carryover	0.000
ID	tij		xij		ki	bg delay	ag delay	yij	delay	q_del
delay 11		10	0		2	-10	53	0	53	0
delay 12		35	0		2	-35	28	0	28	0
delay 13		60	0		2	-60	3	0	3	0
delay 21		15	0		3	-15	48	1	0	0
delay 22		45	0		3	45	18	1	0	0
delay 31		35	0		1	-35.000	28	0	28	0
delay 41		30	0		1	-30.000	33	0	33	0
									145	0
									sum	145.000
			0		0	0	0			
combination 47			0		**0.000**	**0.000**	**0.000**		carryover	0.000
ID	tij		xij		ki	bg delay	ag delay	yij	delay	q_del
delay 11		10	0		2	-10	53	0	53	0
delay 12		35	0		2	-35	28	0	28	0
delay 13		60	0		2	-60	3	0	3	0
delay 21		15	0		3	-15	48	0	48	0
delay 22		45	0		3	45	18	0	18	0
delay 31		35	0		1	-35.000	28	0	28	0
delay 41		30	0		1	-30.000	33	0	33	0
									211	0
									sum	211.000

Appendix 4: Setup for Experiment #3

Experiment #3
Pretimed control - Webster's method
Phase length

$$G_i = \frac{N_i}{\sum_{i=1}^{n} N_i} C$$

Ni, critical flow for phase i	
n, number of phases	
x, saturation flow	1800
l, queue startup avg.	2
R, all-red time	1
C, cycle length	60

Run 1		Run 2		Run 3		Run 4	
n	3	n	3	n	3	n	3
N1	300	N1	600	N1	600	N1	300
N2	900	N2	600	N2	300	N2	1200
N3	600	N3	600	N3	300	N3	300
sum(Ni)	1800	sum(Ni)	1800	sum(Ni)	1200	sum(Ni)	1800
G1	10	G1	20	G1	30	G1	10
G2	30	G2	20	G2	15	G2	40
G3	20	G3	20	G3	15	G3	10

Run 5		Run 6		Run 7		Run 8	
n	5	n	5	n	6	n	6
N1	600	N1	600	N1	300	N1	450
N2	900	N2	600	N2	300	N2	900
N3	300	N3	600	N3	600	N3	450
N4	300	N4	600	N4	300	N4	450
N5	150	N5	300	N5	300	N5	150
sum(Ni)	2250	sum(Ni)	2700	sum(Ni)	1800	sum(Ni)	2400
G1	16	G1	13.33	G1	10	G1	11.25
G2	24	G2	13.33	G2	10	G2	22.5
G3	8	G3	13.33	G3	20	G3	11.25
G4	8	G4	13.33	G4	10	G4	11.25
G5	4	G5	6.667	G5	10	G5	3.75

Experiment #3

Semi-Actuated Control		detector at	100	feet	unit extension	2.27	sec
		speed	30	mph	max. cycle	100	sec

Run 1		Run 2		Run 3		Run 4	
n	3	n	3	n	3	n	3
N1	300	N1	600	N1	600	N1	300
N2	900	N2	600	N2	300	N2	1200
N3	600	N3	600	N3	300	N3	300
sum(Ni)	1800	sum(Ni)	1800	sum(Ni)	1200	sum(Ni)	1800
min1	3	min1	3	min1	3	min1	3
ext1	2.273	ext1	2.273	ext1	2.273	ext1	2.273
max1	40	max1	40	max1	40	max1	40
G2	36	G2	30	G2	30	G2	48
G3	24	G3	30	G3	30	G3	12

Run 5		Run 6		Run 7		Run 8	
n	5	n	5	n	5	n	5
N1	600	N1	600	N1	300	N1	450
N2	900	N2	600	N2	300	N2	900
N3	300	N3	600	N3	600	N3	450
N4	300	N4	600	N4	300	N4	450
N5	150	N5	300	N5	300	N5	150
sum(Ni)	2250	sum(Ni)	2700	sum(Ni)	1800	sum(Ni)	2400
G1	17.14	G1	15	G1	12	G1	12
G2	25.71	G2	15	G2	12	G2	24
G3	8.571	G3	15	G3	12	G3	12
G4	8.571	G4	15	G4	24	G4	12
min5	3	min5	3	min5	3	min5	3
ext5	2.273	ext5	2.273	ext5	2.273	ext5	2.273
max5	40	max5	40	max5	40	max5	40

References

AASHTO, ITE and NEMA. Advanced Transportation Controller (ATC) Standard for the Type 2070 Controller, Version 2.03, March 12, 2004. American Association of State Highway and Transportation Officials, Institute of Transportation Engineers, and National Electrical Manufacturers Association. http://ite.org/standards/atc/ATC2070_2.03.pdf. Accessed July 1, 2005.

Andrews, C.M., S.M. Elahi and J.E. Clark. Evaluation of New Jersey Route 18 OPAC/MIST Traffic-Control System. In Transportation Research Record: Journal of the Transportation Research Board, No. 1603, TRB, National Research Council, Washington, DC, 1998, pp. 150-155.

Box, G.P., G. Jenkins and G. Reisel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, Englewood Cliffs, NJ, 1994.

Chiu, S. and S. Chand. Adaptive Traffic Signal Control Using Fuzzy Logic. Proceedings of the 2nd IEEE International Conference on Fuzzy Systems, San Francisco, March 1993. Institute of Electrical and Electronics Engineers, Piscataway, NJ, 1993.

Coifman, B. Improved Velocity Estimation Using Single Loop Detectors. In Transportation Research Part A Vol. 35. TRB, National Research Council, Washington, DC, 2001. p. 863-880

Coifman, B. Using Dual Loop Speed Traps to Identify Detector Errors. Submitted for presentation and publication at the 1999 Transportation Research Board meeting. http://www-ceg.eng.ohio-state.edu/~coifman/documents/amps.pdf. Accessed July 29, 2005.

Enns, S.T. and L. Li. Optimal Lot-Sizing with Capacity Constraints and Auto-Correlated Interarrival Times. Proceedings of the 2004 Winter Simulation Conference, Washington, DC, December 2004. Institute of Electrical and Electronics Engineers, Piscataway, NJ, 2004.

Fehon, K. Adaptive Traffic Signals: Are We Missing the Boat? Presented at the Institute for Transportation Engineers District 6 2004 Annual Meeting, Sacramento, CA. http://www.dksassociates.com/papers/FehonAdaptiveSignalspaperITE2004b.pdf. Accessed July 19, 2005.

Fletcher, R. *Practical Methods of Optimization*. Wiley, New York, 1987.

Garbacz, R. M. Adaptive Signal Control: What to Expect. Presented at the Institute for Transportation Engineers 2002 Annual Meeting, Philadelphia, PA. ITS Cooperative

Deployment Network, http://209.68.41.108/itslib/AB02H473.pdf. Accessed July 12, 2005.

Gartner, N, F.J. Pooran and C.M. Andrews. Implementation of OPAC Adaptive Control Strategy in a Traffic Signal Network. 2001 IEEE Intelligent Transportation Systems Conference Proceedings, Oakland, CA, August 2001. Institute of Electrical and Electronics Engineers, Piscataway, NJ, 2001.

Gartner, N.H. OPAC: A Demand-Responsive Strategy for Traffic Signal Control. In Transportation Research Record: Journal of the Transportation Research Board, No. 906, TRB, National Research Council, Washington, DC, 1983, pp. 75-81.

Glover, F. Tabu Search – Part 1. In ORSA Journal on Computing, Vol. 1, No. 3. Operation Research Society of America, 1989.

Greenough, J.C. and W.L. Kelman. Metro Toronto SCOOT: Traffic Adaptive Control Operation. ITE Journal, Vol. 58, No. 5, May 1998.

Greenshields, B.D., Shapiro, D. and Ericksen, E.L. Traffic Performance at Urban Intersections. Technique Report No. 1. Bureau of Highway Traffic, Yale University, New Haven, CT, 1947.

Hamed, M.H., H.R. Al-Masaeid, and Z.M. Bani Said. Short-Term Prediction of Traffic Volume in Urban Arterials. In the Journal of Transportation Engineering, Vol. 121, No. 3, 1995. American Society of Civil Engineers, Reston, VA, 1995.

Herman, R. and I. Prigogine. A Two-Fluid Approach to Town Traffic. In Science, New Series, Vol. 204, No. 4389, 1979, pp. 148-151.

Hernandez, J., J. Cuena, , and M. Molina. Real-time Traffic Management through Knowledge-based Models: The TRYS Approach. In ERUDIT Tutorial on Intelligent Traffic Management Models. ERUDIT, Helsinki, Finland, 1999.

Hobeika, A.G. and C. Kim. Traffic-Flow Prediction Systems Based on Upstream Traffic. 1994 Vehicle Navigation and Information Systems Conference Proceedings, Yokohama, Japan, August-September 1994. Institute of Electrical and Electronics Engineers, Piscataway, NJ, 2001.

Ishak, S. and H. Al-Deek. Performance Evaluation of Short-Term Time-Series Traffic Prediction Model. In the Journal of Transportation Engineering, Vol. 128, No. 6, 2002. American Society of Civil Engineers, Reston, VA, 2002.

ITT Industries, Inc. Systems Division. *CORSIM Run-Time Extension (RTE) Developer's Guide*, Version 5.1. Federal Highway Administration, McLean, VA, 2003.

Jhaveri, C.S., J. Perrin and P.T. Martin. SCOOT: An Evaluation of Its Effectiveness Over a Range of Congestion Intensities. Presented at the 82nd Annual Meeting of the Transportation Research Board, Washington, D.C., January 2003. TRB, National Research Council, Washington, DC, 2003.

Kluwer's International Series. *Handbook of Metaheuristics*. (Glover, F. and G. Kochenberger, ed.) Kluwer Academic Publishers, Boston, MA, 2003.

Laguna, M., J.W. Barnes and F. Glover. Tabu search methods for a single machine scheduling problem. In the Journal of Intelligent Manufacturing, Vol. 2, pp. 63-74. Chapman and Hall, 1991.

Lin, F. and D.J. Cooke. Potential Performance Characteristics of Adaptive Control at Individual Intersections. In Transportation Research Record: Journal of the Transportation Research Board, No. 1057, TRB, National Research Council, Washington, DC, 1986, pp. 30-32.

Liu, H.X., J. Oh and W. Recker. Adaptive Signal Control System with On-Line Performance Measure. Paper No. UCI-ITS-WP-01-12. Institute of Transportation Studies, University of California – Irvine, Irvine, CA, 2001.

Lowrie, P. SCATS – The History of Its Development. Presented at the 8th World Congress on Intelligent Transportation Systems, Sydney, Australia, 2001.

May, A.D. *Traffic Flow Fundamentals*. Prentice Hall, Englewood Cliffs, NJ, 1990.

Mirchandani, P. and L. Head. A Real-Time Traffic Signal Control System: Architecture, Algorithms and Analysis. In Transportation Research Part C Vol. 9, No. 6. TRB, National Research Council, Washington, DC, 2001, pp. 415-432.

Oh, C. and S.G. Ritchie. Real-Time Inductive-Signature Based Level of Service for Signalized intersections. Paper No. UCI-ITS-WP-01-10. Institute of Transportation Studies, University of California – Irvine, Irvine, CA, 2001.

Oh, S., S.G. Ritchie and C. Oh. Real Time Traffic Measurement from Single Loop Inductive Signatures. Paper No. UCI-ITS-WP-01-15. Institute of Transportation Studies, University of California – Irvine, Irvine, CA, 2001.

Panda, D.P. and C.A. Anderson. Low Cost Remote Management of Video Detectors. Presented at the 6th World Congress on ITS, Toronto, Canada, November 1999. http://www.autoscope.com/papers/Low%20Cost%20Remote%20Management%20of%20Video%20Detectors.pdf. Accessed July 8, 2005.

Petty, K.F., P. Bickel, M. Ostland, J. Rice, Y. Ritov and F. Schoenberg. Accurate Estimation of Travel Times from Single-Loop Detectors. In Transportation Research, Part A Vol. 32A, No.1. TRB, National Research Council, Washington, DC, 1998. p. 1-17

Pignataro, L.J. *Traffic Engineering: Theory and Practice*. Prentice Hall, Englewood Cliffs, NJ, 1973.

Porche, I. and S. Lafortune. Adaptive Look-Ahead Optimization of Traffic Signals. Technical Report CGR 97-11. Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 1997.

Roozemond, D. A. and J. L. H. Rogier. Agent Controlled Traffic Lights. Proceedings of ESIT 2000: European Symposium on Intelligent Techniques, Aachen, Germany, September 2000. ERUDIT, Helsinki, Finland, 2000.

Texas Transportation Institute. PASSER II-02 User's Manual. Center for Microcomputers in Transportation, University of Florida, Gainesville, FL, 2003.

Tomer, E., L. Safonov, N. Madar, S. Havlin. Optimization of Congested Traffic Flow in Systems With a Localized Periodic Inhomogeneity. Dated June 11, 2005. http://arxiv.org/PS_cache/cond-mat/pdf/0105/0105493.pdf. Accessed July 29, 2005.

Transportation Research Center, University of Florida. TRANSYT-7F User's Manual (Release 6). Center for Microcomputers in Transportation, University of Florida, Gainesville, FL, 1988.

Van Arem, B., H.R. Kirby, M.J.M. Van Der Vlist and J.C. Whittaker. Recent Advances and Applications in the Field of Short-Term Traffic Prediction. In the International Journal of Forecasting, Vol. 13 (1997). Elsevier, 1997.

Vita

Michael Shenoda was born April 23, 1977 in Staten Island, New York to Shenoda Tadros Shenoda and Violette Asham-Shenoda. He received a Bachelor of Science degree in Civil Engineering in January 1998 and a Master of Science degree in Environmental Engineering in August 1999, both from the New Jersey Institute of Technology in Newark, New Jersey. His previous experience in the field of transportation includes one year as a Junior Engineer with the New York State Department of Transportation in Queens, New York, two years as a Highway Design Engineer with HNTB Corporation in Fairfield, New Jersey, and one year as a Highway Designer with STV Incorporated in Trenton, New Jersey. He is a registered Professional Engineer in State of New Jersey, with inactive licensure in the State of Texas.

Permanent address: 2907 Rio Grande Street, Austin, Texas 78705

This dissertation was typed by the author.